11-2023

Convolutional Neural Network-Based Gene Prediction Using Buffalograss as a Model System

Michael Morikone
University of Nebraska-Lincoln

CONVOLUTIONAL NEURAL NETWORK-BASED GENE PREDICTION USING BUFFALOGRASS AS A MODEL SYSTEM

by

Michael Morikone

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Complex Biosystems

(Systems Analysis)

Under the Supervision of Professor Keenan Amundsen

Lincoln, Nebraska

November, 2023

CONVOLUTIONAL NEURAL NETWORK-BASED GENE PREDICTION USING BUFFALOGRASS AS A MODEL SYSTEM

Michael Morikone, Ph.D.

University of Nebraska, 2023

Advisor: Keenan Amundsen

The task of gene prediction has been largely stagnant in algorithmic improvements compared to when algorithms were first developed for predicting genes thirty years ago. Rather than iteratively improving the underlying algorithms in gene prediction tools by utilizing better performing models, most current approaches update existing tools through incorporating increasing amounts of extrinsic data to improve gene prediction performance. The traditional method of predicting genes is done using Hidden Markov Models (HMMs). These HMMs are constrained by having strict assumptions made about the independence of genes that do not always hold true. To address this, a Convolutional Neural Network (CNN) based gene prediction tool was developed and named GeneCNN. Due to their nonlinearity, neural networks are adept at capturing complex relationships between data points when applied to sufficiently large datasets such as whole genomes. Convolutional neural networks further improve upon neural networks through the incorporation of spatial dependence into individual datapoints. GeneCNN was trained using a sequenced buffalograss (*Bouteloua dactyloides*) genome. Training performance of GeneCNN resulted in a 97% accuracy in correctly identifying genic sequences in test data. GeneCNN uniquely identified a greater number of genes than currently existing gene prediction tools BRAKER3, AUGUSTUS, and Fgenesh at 1,089, 1,535, and 478

respectively, when using a 10 million nucleotide length genome sequence of buffalograss as input. Gene predictions made by combinations of the tools BRAKER3, AUGUSTUS, and Fgenesh, were compared to GeneCNN to assess the percentage of gene predictions made by GeneCNN that are supported by at least one other tool, where support ranged from 40.5% to 84.1% of all GeneCNN gene predictions for every combination of BRAKER3, AUGUSTUS, and Fgenesh. The findings in this study support the use of CNNs for gene prediction and serve as a valuable resource for the improvement of gene prediction algorithms in future research.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Keenan Amundsen for his guidance and patience. Through his mentorship I have grown both as a scholar and as a person, for which I am eternally grateful. I would also like to thank my committee members, Dr. Juan Cui, Dr. Gautam Sarath, and Dr. Chi Zhang, for their support in my academic endeavor as they provided valuable insight throughout my project. I also want to thank our collaborators, the Huff lab at Pennsylvania State University. Without Dr. David Huff and his former student Chris Benson, we would not have had a buffalograss genome for this project. I am thankful to the Complex Biosystems program, especially Dr. Jennifer Clarke who has been helpful and understanding throughout my time here. I am also thankful for the Holland Computing Center as without their resources this project would not have been computationally feasible.

I also want to thank the professors who mentored me during my undergraduate career and grew my interest in science, especially Dr. David Rhoads, Dr. Jeremy Dodsworth, Dr. Arturo Concepcion, and Dr. Christopher Wilson.

Lastly, I would like to thank my friends and family, especially Alex, whose support has been valuable throughout my time in Nebraska. Without the support of everyone I would not be where I am today.

GRANT INFORMATION

This study was supported by grants awarded to Dr. Keenan Amundsen through the United States Golf Association with project IDs 2023-11-778 and 2021-04-728.

TABLE OF CONTENTS

LIST OF TABLES……..ix

LIST OF FIGURES…….x

CHAPTER 1: INTRODUCTION……1

1.1 Gene prediction……..1

1.1.1 Similarity-based gene prediction……2

1.1.2 Ab initio gene prediction…….....2

1.2 Algorithms and models commonly used in gene prediction……..3

1.2.1 Dynamic programming……...4

1.2.2 Hidden Markov Models……..6

1.2.3 Neural networks……..8

1.3 Improving *ab initio* gene prediction with neural networks……..10

1.4 The necessity for improvement in gene prediction……..11

References……..12

CHAPTER 2: DEVELOPMENT OF GENECNN……15

2.1 Convolutional neural networks……15

2.1.1 Data processing for CNNs……18

2.1.2 Activation functions and optimization algorithms……19

2.1.3 CNN overfitting and regularization……..20

2.1.4 Hyperparameter optimization……...21

2.2 Measuring the performance of CNNs……..22

2.3 Developing GeneCNN for gene prediction……..24

Methods……..25

2.4 Preparation of buffalograss genome by Dovetail Genomics……...25

2.5 Initial annotation with BRAKER2……...25

2.6 Removal of contaminants and repeat elements from the buffalograss genome……...26

2.7 GeneCNN development……...28

2.8 GeneCNN performance improvements……30

Results……31

2.9 Initial annotation with BRAKER2……...31

2.10 Removal of contaminants and repeat elements from the buffalograss genome……...31

2.11 GeneCNN development……32

2.12 GeneCNN performance……36

Discussion……..40

References……..44

CHAPTER 3: ASSESSMENT AND VALIDATION OF GENECNN……47

3.1 Predicting genes with GeneCNN……48

3.1.1 Gene prediction workflow……48

3.2 AUGUSTUS……52

3.3 BRAKER3……...53

3.4 Fgenesh……54

3.5 Comparing GeneCNN with other gene prediction tools……..……...…….55

Methods……..57

3.6 Creation of a dataset for gene prediction with GeneCNN……...57

3.7 Creation of a sliding window for GeneCNN predictions……57

3.8 Defining GeneCNN gene predictions……..58

3.9 Running BRAKER3……59

3.10 Running AUGUSTUS……...60

3.11 Running Fgenesh……...60

3.12 Creating total predicted gene counts……..60

Results……61

3.13 GeneCNN gene predictions……...…61

3.14 BRAKER3, AUGUSTUS, and Fgenesh gene predictions…….64

3.15 Comparison of gene predictions with all tools……..64

Discussion……..69

References……..73
APPENDIX A: SUPPLEMENTAL FIGURES…….76

LIST OF TABLES

Table 2-1. Final GeneCNN hyperparameters for the first major version of the gene prediction tool……34

Table 2-2. GeneCNN performance across different methods during development……..37

Table 3-1. Uniquely predicted genes from GeneCNN compared to BRAKER3, AUGUSTUS, and Fgenesh……62

Table 1-2. Uniquely predicted genes from BRAKER3, AUGUSTUS, and Fgenesh, compared to GeneCNN……..65

Table 3-3. The number of gene predictions made by GeneCNN that are supported by BRAKER3, AUGUSTUS, Fgenesh, or some combination of the three……66

Table 3-4. The number of gene predictions made by BRAKER3 that are supported by AUGUSTUS, Fgenesh, or the combination of AUGUSTUS and Fgenesh……...67

Table 3-5. The number of gene predictions made by AUGUSTUS that are supported by BRAKER3, Fgenesh, or the combination of BRAKER3 and Fgenesh…….67

Table 3-6. The number of gene predictions made by Fgenesh that are supported by BRAKER3, AUGUSTUS, or the combination of BRAKER3 and AUGUSTUS……..…68

LIST OF FIGURES

Figure 2-1. A generalized visual representation of GeneCNN……17

Figure 2-2. GeneCNN layout in sequential order of layers and operations as output from Tensorflow plot_model……35

Figure 2-3. A graphical display of the model loss over time using the current iteration of GeneCNN……39

Figure 2-4. A graphical display of the model accuracy over time using the current iteration of GeneCNN……40

Figure 3-1. Visualization of the GeneCNN gene prediction workflow……51

Figure A-1. Historical graph result of the model loss over time using the '200 nt' iteration of GeneCNN……76

Figure A-2. Historical graph result of the model accuracy over time using the '200 nt' iteration of GeneCNN……77

Figure A-3. Historical confusion matrix for the '200 nt' iteration of GeneCNN……78

Figure A-4. Historical graph result of the model loss over time using the '1 Conv Layer' iteration of GeneCNN……79

Figure A-5. Historical graph result of the model accuracy over time using the '1 Conv Layer' iteration of GeneCNN……80

Figure A-6. Historical confusion matrix for the '1 Conv Layer' iteration of GeneCNN……81

Figure A-7. Historical graph result of the model loss over time using the '4.9:1 Class Ratio' iteration of GeneCNN……82

Figure A-8. Historical graph result of the model accuracy over time using the '4.9:1 Class Ratio' iteration of GeneCNN……83

Figure A-9. Historical confusion matrix for the '4.9:1 Class Ratio' iteration of GeneCNN……84

Figure A-10. Historical graph result of the model loss over time using the 'Manual Adjust' iteration of GeneCNN……85

Figure A-11. Historical graph result of the model accuracy over time using the 'Manual Adjust' iteration of GeneCNN……86

Figure A-12. Historical confusion matrix for the 'Manual Adjust' iteration of GeneCNN……87

Figure A-13. Historical graph result of the model loss over time using the ‘No BN’ iteration of GeneCNN……....88

Figure A-14. Historical graph result of the model accuracy over time using the ‘No BN’ iteration of GeneCNN……89

Figure A-15. Historical confusion matrix for the ‘No BN’ iteration of GeneCNN……...90

CHAPTER 1: INTRODUCTION

Buffalograss [*Bouteloua dactyloides* (Nutt.) Columbus] is a warm-season prairie grass found natively in the United States Great Plains region. Buffalograss is highly tolerant to drought and heat; it also tolerates mowing and forms sod, making it a valuable turfgrass species (Steinke et al., 2010). As a non-model organism, buffalograss lacks many of the genomic resources available to model organisms, such as a reference genome. The use of a reference genome for guiding sequence analysis generates higher quality results, allowing for discoveries that would not likely be made with a *de novo* approach alone (Marchant et al., 2014). However, a reference genome alone is not enough, as with the explosion of genomic sequencing and annotation, gene prediction quality has suffered due to the propagation of errors within databases, lowering the quality of similarity-based gene prediction (Salzberg, 2019). In addition to the errors associated with similarity-based gene prediction, *ab initio* gene prediction also has lower accuracy than potentially achievable due to traditional methods incorrectly applying statistical models. The focus of this project has been on creating a robust reference genome for buffalograss by refining gene prediction through the use of deep learning models rather than relying on inappropriate traditional gene prediction methods.

1.1 Gene prediction

Gene prediction is a complex task in eukaryotes that has seen relatively few algorithmic improvements in recent years compared to the inception of computational tools fit for this task. The main difficulties of eukaryotic gene prediction are due to low genic content relative to the whole genome and the presence of introns, repetitive elements, and conserved domains within genes having different functions. This has led to

two major approaches for gene prediction, similarity-based gene prediction and *ab initio* gene prediction.

1.1.1 Similarity-based gene prediction

Similarity-based gene prediction is done through the use of extrinsic data, typically with known genomic DNA or cDNA aligned against the unknown sequences of interest. This alignment with known sequences has shown to be particularly adept at identifying exons if the aligned cDNA is from either the same or a closely related organism (Fukunishi et al., 1999). One of the advantages of using known extrinsic data is that sequence similarity can be used to identify homology of known sequences to the unknown sequences of interest, allowing for inference of the unknown sequence function. Sequence similarity suggests common ancestry, and therefore likely similar function. This identification of similarity allows for identification of potential function of previously uncharacterized genes (Pearson, 2013). As some gene functions are experimentally verified, especially in well-studied systems such as model organisms, this method gives some credibility to the newly informed gene function. However, a major downside of similarity-based gene prediction is that unless the genes in question are highly conserved, this method is limited by how similar two organisms are, as well as by the number and quality of gene sequences that are currently in databases.

1.1.2 *Ab initio* gene prediction

The other major gene prediction approach, *ab initio* gene prediction, is an intrinsic method which works by utilizing information from sequence motifs in conjunction with statistical-based models such as Hidden Markov Models (HMMs) and Neural Networks

(NNs) or other approaches such as Dynamic Programming (DP) (Wang et al., 2004). There are two broad categories of sequence information used for *ab initio gene* prediction, including signal sensors and content sensors. Signal sensors are composed of different types of sequence motifs such as start and stop codons, splice sites, transcription start sites, and polyadenylation. Content sensors instead utilize statistical properties of genomic regions, such as sequence composition or length, oligomer frequency, and 3-base periodicity to identify a genomic region as coding or non-coding (Do & Choi, 2005). Coding measures, or the likelihood of a given sequence being protein coding, have been one of the foundations of gene prediction algorithms, which include the previously mentioned content sensors. Fickett and Tung (1992) found that of these coding measures, in-phase hexamer count was the most accurate for providing discrimination between coding and non-coding regions. They also found that it is a best practice to both perform coding measures and decisions on discrimination in the same algorithm, which is a method performed by neural networks.

1.2 Algorithms and models commonly used in gene prediction

Algorithms and models used for gene prediction started with programming techniques such as DP and then moved on to machine learning methods such as HMMs, which are statistical models used for the extraction of essential information (Seymore et al., 1999). The field of machine learning is characterized by information extraction and adaptation based on information gleaned from data. In the scope of gene prediction, the application of machine learning algorithms is in the classification of sequences as genes, where the focus is on the accuracy of correctly identifying genes given a specific algorithm and dataset size (Jordan & Mitchell, 2015).

Deep learning is a type of machine learning, which has been popularized recently due to its higher potential accuracy when compared against machine learning methods. Deep learning is notable for its use of models containing nonlinear transforming layers, which allow these models to better extract information from complex datasets such as from genomic data when compared to machine learning methods. Neural networks, of this area of deep learning, have seen use in gene prediction but have not achieved the same popularity as HMMs due to their higher degree of difficulty for implementation and interpretation of results.

1.2.1 Dynamic programming

Dynamic programming has been one of the most prolific programming methods in bioinformatics and has been used for approaches such as sequence alignment, gene prediction, and predicting the secondary structure of functional RNA (Giegerich, 2000). This programming method is used when a problem has an optimal substructure, where the problem can be split into optimal subproblems that are able to be solved recursively, allowing a given solution to be found by iteratively solving the simpler subproblems. The defining factor of having optimal solutions to subproblems allows for the overall solution to be solved optimally, making DP methods efficient by running in polynomial time (Xu & Wu, 2021).

As DP is solely an optimization method for solving programming problems and not a type of statistical model, DP fits into similarity-based or a combination-based approach by adding extrinsic data for support to *ab initio* gene prediction approaches. As such, DP is not directly used for predicting genes but rather for methods such as sequence alignment and gene structure assembly. These methods are often used alongside

statistical models with their signal and content sensors to predict genes optimally and accurately (Howe et al., 2002).

Similarity-based gene prediction with a DP approach was largely unexplored when algorithmic gene prediction methods were first being developed due to the combinatorial magnitude of search space, as a native implementation of DP needs to examine all potential exon assemblies given a genomic sequence and a known target protein. This initial problem of a combinatorial search space was addressed by creating a spliced alignment algorithm where instead of examining all potential exon assemblies, only the exon assembly most related to a known target protein was retained, thereby reducing the search space and making this a sparse DP problem (Gelfand et al., 1996). Sparse DP is a method by which rather than considering every potential entry in a DP matrix, only a subset is considered that has the largest degree of importance for a given problem (Baker & Giancarlo, 1999). This use of sparse DP made the combinatorial problem encountered by Gelfand et al. (1996) run in polynomial time, making the previously infeasible problem with DP optimized. Another method for similarity-based gene prediction using DP was developed by narrowing the search space with strict requirements of containing start/stop codons, consensus sequences, and consistent assigned exon frames (Pachter et al., 1999). This narrowing of search space allowed the algorithm to utilize ever growing databases of proteins and annotated genes for a similarity-based gene prediction approach. With these two different methods developed, it has been shown that while DP is not a popular approach for gene prediction on its own, it is feasible.

The more commonly used method of incorporating DP into gene prediction is by using it in conjunction with statistical models. One such approach was taken by VEIL, an HMM-based system that was first trained on a given set of eukaryotic DNA sequences to find coding exons, then a DP algorithm was used to parse a new sequence found from the HMM results into its corresponding exons and introns (Henderson, 1997). This addition of DP to an HMM-based gene predictor allowed for finding the optimal gene models rather than solely using output from the HMM.

1.2.2 Hidden Markov Models

Like DP, HMMs have seen much use in the field of bioinformatics, where they have been the most widely used models for *ab initio* gene prediction. These HMMs are statistical models where unobservable states exist with distinct probabilities of transitioning between states, as well as observable sequences which are emitted from this set of state transitions (Eddy, 2004). In gene prediction these observable outputs are sequences of nucleotides, which give statistical significance towards identifying genes. Hidden Markov Models are defined by a set number of states contained within the model, a set number of distinct observations output per state, the probability of transitioning between states, the probability of a distinct observation at any given state, and the initial distribution of states (Rabiner, 1989). Given these parameters it is possible to create a robust stochastic model which provides an output of sequences determined from the probability of state transitions.

One commonly used approach for eukaryotic genome annotation is BRAKER2, which utilizes GeneMark-ET and AUGUSTUS, both of which use HMMs for gene prediction (Brůna et al., 2021; Lomsadze et al., 2014; Stanke et al., 2008). Both of these

tools have had updates since their original creation in order to utilize extrinsic data, namely unassembled transcriptomic reads. Utilization of this extrinsic evidence supports the *ab initio* gene prediction algorithms by changing the training from unsupervised to semi-supervised, meaning that some of the data are now labeled as being genic. This addition of external support to predictions made by the HMM-based algorithms has greatly improved accuracy, with testing for GeneMark-ET showing that sensitivity and specificity were increased by 24.5% in *Aedes aegypti* (Lomsadze et al., 2014).

The path of states within HMMs is a Markov chain, which assumes that any given observation is solely dependent on its immediately prior state while independent of all other prior states (Henderson et al., 1997). However within biology, gene order and chromosomal location are highly conserved through natural selection in evolutionarily related organisms which is referred to as synteny, therefore this primary Markovian assumption of independence is flawed when applied to gene prediction (Renwick, 1971). It has been shown previously in the field of natural language processing that HMMs implemented with these flawed assumptions can still yield satisfactory results, though nonoptimal (Blunsom, 2004). The satisfactory results combined with the relative ease of implementation and computational simplicity compared to other statistical models has led to HMMs being readily used despite scenarios where the foundational assumption of Markovian independence is flawed. This relative ease of modeling with HMMs has led to their acceptance as the most common gene prediction models despite their nonoptimal results, as extrinsic data has been increasingly used to improve results, similar to DP. Accuracy in *ab initio* gene prediction would improve by utilizing other types of statistical

models that do not break foundational assumptions, one such category of model is that of neural networks.

1.2.3 Neural networks

While deep learning is a branch of machine learning, it is in many ways also an extension of the foundational ideas in machine learning. Rather than requiring careful data preprocessing or feature extraction traditionally found in machine learning, deep learning can be applied to raw or minimally preprocessed data and automatically discover the representation needed to classify any given data point (LeCun et al., 2015). Additionally, deep learning models are characterized by the extraction of high-level features through successive layers of learning on input data (Najafabadi et al., 2015). Given these aspects of deep learning models, they appear to be well-fit to replace HMMs in gene prediction.

Deep learning generally outperforms machine learning methods with proper model building and parameter tuning given its nonlinear nature and strength of learning on large datasets (Korotcov et al., 2017). For example, in a study done by Ni et al. (2019), it was shown that a deep learning approach outperforms an HMM approach for detecting DNA methylation by 7-30% for accuracy when measured on human data. Cheng et al. (2016) found that their deep learning approach for predicting miRNA targets outperformed machine learning methods by 15-40%. These two examples suggest that there is value in moving from the traditional HMM approach to deep learning for gene prediction.

Deep learning revolves around neural networks which simulate the human brain by allowing for information transfer between nodes called neurons. To do this, basic neural networks are composed of several layers of nodes, specifically an input layer, at minimum one hidden layer, and an output layer. Depending on defined thresholds, data above these thresholds are transferred between layers until reaching the output layer, which in the case of this project is classification of any particular data point as genic or nongenic. This basic form of deep learning, otherwise known as a feedforward neural network, is the foundation for deep learning as a whole. The term feedforward arose from the flow of information from input to output, while the term deep learning is related to the depth of the model, otherwise known as having multiple hidden layers found within a neural network.

The hidden layers of a neural network are where approximations of the prior input happen to closely mimic the output in a way that is obscured to the viewer, thus the "hidden layer" name. In order to mimic the output, nonlinear functions are applied to the input to transform it into a close approximation (Goodfellow et al., 2016). This nonlinearity allows for neural networks to outperform machine learning models, as neural networks can more readily capture essential features from complex patterns (Mohseni-Dargah et al., 2022).

A large part of creating a good approximation of the expected output in deep learning is in parameterization, where the optimizer and cost functions are chosen. The optimizer of a neural network is an algorithm or function that is used for changing the weights associated with data as well as the learning rate in order to reduce loss, which is the difference between the expected output and the current output. Cost functions are an

extension of this metric of loss by then measuring the loss over the entirety of the dataset, giving a value to the performance of the neural network when applied to a dataset. Another important consideration when applying neural networks is the choice of an activation function, which is used for computation in hidden layers to apply nonlinearity and transform the data.

1.3 Improving *ab initio* gene prediction with neural networks

As shown previously, there are opportunities for improvement in sequence analysis models through the use of NNs as opposed to the commonly used HMMs. While NNs are not as popular in traditional gene prediction when compared to HMMs, NNs are not a new area of study. Uberbacher and Mural (1991) first described GRAIL, a NN-based gene predictor, more than twenty years ago. GRAIL was trained on output from seven different sensor algorithms, each of which providing information on the coding potential of a given sequence, allowing for final predictions to be made on the test data for coding DNA locations. Similar to GRAIL, GeneParser was another early tool that used NNs for gene prediction (Snyder & Stormo, 1992). GeneParser trained their NN using weights developed from five different exon content classifications. Snyder and Stormo (1992) demonstrated that both GRAIL and GeneParser outperformed GeneID, another prominent gene predictor at the time that avoided statistical models and instead utilized a hierarchical rule system for filtering and predicting genes (Guigó et al., 1992).

Though NNs themselves are not new to gene prediction, Convolutional Neural Networks (CNNs) have yet to be explored for this purpose. Convolutional neural networks are one form of NNs that are useful in working with spatially dependent data. As genomes not only have interactions among coding and noncoding regions, but also

have dependence of genic regions on other regions such as promoters, CNNs appear to be a suitable method for predicting genes. To resolve the spatial nature of data, CNNs first produce output from their neurons by taking a weighted sum across the length of the data using a kernel that scans the entire input in a convolutional layer. This output is then used as input for pooling layers which summarize the input data after nonlinear functions are applied, allowing for a dimensionality reduction while also keeping the important aspects of the data (Krizhevsky et al., 2017). This successive process of convolution and pooling can also be repeated to find motifs in the data, making CNNs adept at finding patterns, which would be useful for *ab initio* gene prediction (Umarov & Solovyev, 2017).

1.4 The necessity for improvement in gene prediction

The majority of current gene predictors are no longer solely similarity-based or *ab initio*, but rather some combination of the two. By combining extrinsic evidence with statistical models, the accuracy of gene predictions has greatly improved. However, *ab initio* prediction methods still struggle on their own with atypical cases, such as with small proteins or organism-specific genes (Scalzitti et al., 2020). Rather than constantly striving to improve prediction algorithms like studies in the past, the current trend has been stagnant on HMMs and the incorporation of extrinsic evidence. Whether it be solely to detach from HMMs to avoid the broken Markovian assumption of independence, or for general improvements seen with NNs especially when leveraging large datasets, it appears to be in the field's best interest to refocus on improving *ab initio* gene prediction algorithms. The use of CNNs to strive towards this is recommended here due to their promise in reducing complex spatially dependent datasets to their essential features.

References

Baker, B. S., & Giancarlo, R. (2002). Sparse dynamic programming for longest common subsequence from fragments. *Journal of Algorithms*, *42*(2), 231–254. https://doi.org/10.1006/jagm.2002.1214

Blunsom, P. (2004). Hidden Markov Models.

Brůna, T., Hoff, K. J., Lomsadze, A., Stanke, M., & Borodovsky, M. (2021). BRAKER2: Automatic eukaryotic genome annotation with GeneMark-EP+ and AUGUSTUS supported by a protein database. *NAR Genomics and Bioinformatics*, *3*(1). https://doi.org/10.1093/nargab/lqaa108

Cheng, S., Guo, M., Wang, C., Liu, X., Liu, Y., & Wu, X. (2016). MiRTDL: A deep learning approach for miRNA target prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *13*(6), 1161–1169. https://doi.org/10.1109/TCBB.2015.2510002

Do, J. H., & Choi, D. K. (2006). Computational approaches to gene prediction. *Korean Journal of Microbiology*, *44*(2), 137–144.

Eddy, S. R. (2004). What is a hidden Markov model? *Nature Biotechnology*, *22*, 1315–1316. https://doi.org/10.1038/nbt1004-1315

Fickett, J. W., & Tung, C. S. (1992). Assessment of protein coding measures. *Nucleic Acids Research*, *20*(24), 6441–6450. https://doi.org/10.1093/nar/20.24.6441

Fukunishi, Y., Suzuki, H., Yoshino, M., Konno, H., & Hayashizaki, Y. (1999). Prediction of human cDNA from its homologous mouse full-length cDNA and human shotgun database. *FEBS Letters*, *464*(3), 129–132. https://doi.org/10.1016/S0014-5793(99)01696-8

Gelfand, M. S., Mironov, A. A., & Pevzner, P. A. (1996). Gene recognition via spliced sequence alignment. *Proceedings of the National Academy of Sciences of the United States of America*, *93*(17), 9061–9066. https://doi.org/10.1073/pnas.93.17.9061

Giegerich, R. (2000). A systematic approach to dynamic programming in bioinformatics. *Bioinformatics*, *16*(8), 665–677. https://doi.org/10.1093/bioinformatics/16.8.665

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Guigó, R., Knudsen, S., Drake, N., & Smith, T. (1992). Prediction of gene structure. *Journal of Molecular Biology*, *226*(1), 141–157. https://doi.org/10.1016/0022-2836(92)90130-C

Henderson, J., Salzberg, S., & Fasman, K. H. (1997). Finding genes in DNA with a hidden Markov model. *Journal of Computational Biology*, *4*(2), 127–141. https://doi.org/10.1089/cmb.1997.4.127

Howe, K. L., Chothia, T., & Durbin, R. (2002). GAZE: A genetic framework for the integration of gene-prediction data by dynamic programming. *Genome Research*, *12*(9), 1418–1427. https://doi.org/10.1101/gr.149502

Jordan, M., & Mitchell, T. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260. https://doi.org/10.1126/science.aac4520

Korotcov, A., Tkachenko, V., Russo, D. P., & Ekins, S. (2017). Comparison of deep learning with multiple machine learning methods and metrics using diverse drug discovery data sets. *Molecular Pharmaceutics*, *14*(12), 4462–4475. https://doi.org/10.1021/acs.molpharmaceut.7b00578

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444. https://doi.org/10.1038/nature14539

Lomsadze, A., Burns, P. D., & Borodovsky, M. (2014). Integration of mapped RNA-Seq reads into automatic training of eukaryotic gene finding algorithm. *Nucleic Acids Research*, *42*(15). https://doi.org/10.1093/nar/gku557

Marchant, A., Mougel, F., Mendonça, V., Quartier, M., Jacquin-Joly, E., da Rosa, J. A., Petit, E., & Harry, M. (2015). Comparing de novo and reference-based transcriptome assembly strategies by applying them to the blood-sucking bug *Rhodnius prolixus*. *Insect Biochemistry and Molecular Biology*, *69*, 25-33. https://doi.org/10.1016/j.ibmb.2015.05.009

Mohseni-Dargah, M., Falahati, Z., Dabirmanesh, B., Nasrollahi, P., & Khajeh, K. (2022). Machine learning in surface plasmon resonance for environmental monitoring. *Artificial Intelligence and Data Science in Environmental Sensing*, 269–298. https://doi.org/10.1016/B978-0-323-90508-4.00012-5

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, *2*(1), 1-21. https://doi.org/10.1186/s40537-014-0007-7

Ni, P., Huang, N., Zhang, Z., Wang, D. P., Liang, F., Miao, Y., Xiao, C. le, Luo, F., & Wang, J. (2019). DeepSignal: Detecting DNA methylation state from Nanopore sequencing reads using deep-learning. *Bioinformatics*, *35*(22), 4586–4595. https://doi.org/10.1093/bioinformatics/btz276

Pachter, L., Batzoglou, S., Spitkovsky, V. I., Banks, E., Lander, E. S., Kleitman, D. J., & Berger, B. (1999). A dictionary-based approach for gene annotation. *Journal of Computational Biology*, *6*(3), 419–430.

Pearson, W. R. (2013). An introduction to sequence similarity ("homology") searching. *Current Protocols in Bioinformatics*, *42*(1), 3.1.1-3.1.8. https://doi.org/10.1002/0471250953.bi0301s42

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286. https://doi.org/10.1109/5.18626

Renwick, J. H. (1971). The mapping of human chromosomes. *Annual Review of Genetics*, *5*(1), 81–120. https://doi.org/10.1146/annurev.ge.05.120171.000501

Salzberg, S. L. (2019). Next-generation genome annotation: We still struggle to get it right. *Genome Biology*, *20*(92). https://doi.org/10.1186/s13059-019-1715-2

Scalzitti, N., Jeannin-Girardon, A., Collet, P., Poch, O., & Thompson, J. D. (2020). A benchmark study of ab initio gene prediction methods in diverse eukaryotic organisms. *BMC Genomics*, *21*(293), 1–20. https://doi.org/10.1186/s12864-020-6707-9

Seymore, K., Mccallum, A., & Rosenfeld, R. (1999). Learning hidden Markov model structure for information extraction.

Snyder, E. E., & Stormo, G. D. (1993). Identification of coding regions in genomic DNA sequences: An application of dynamic programming and neural networks. *Nucleic Acids Research*, *21*(3), 607–613. https://doi.org/10.1093/nar/21.3.607

Stanke, M., Diekhans, M., Baertsch, R., & Haussler, D. (2008). Using native and syntenically mapped cDNA alignments to improve de novo gene finding. *Bioinformatics*, *24*(5), 637–644. https://doi.org/10.1093/bioinformatics/btn013

Steinke, K., Chalmers, D., Thomas, J., White, R., Lansing, E., & Chalmers, D. (2011). Bermudagrass and Buffalograss drought response and recovery at two soil depths. *Crop Science*, *51*(3), 1215–1223. https://doi.org/10.2135/cropsci2010.08.0469

Uberbacher, E. C., & Mural, R. J. (1991). Locating protein-coding regions in human DNA sequences by a multiple sensor-neural network approach. *Proceedings of the National Academy of Sciences of the United States of America*, *88*(24), 11261–11265. https://doi.org/10.1073/pnas.88.24.11261

Umarov, R. K., & Solovyev, V. V. (2017). Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. *PLoS ONE*, *12*(2). https://doi.org/10.1371/journal.pone.0171410

Wang, Z., Chen, Y., & Li, Y. (2004). A brief review of computational gene prediction methods. *Genomics, Proteomics & Bioinformatics*, *2*(4), 216–221. https://doi.org/10.1016/S1672-0229(04)02028-5

Xu, J., & Wu, S. (2021). Analysis and application of dynamic programming. *Journal of Physics: Conference Series*, *1865*(4). https://doi.org/10.1088/1742-6596/1865/4/042023

CHAPTER 2: DEVELOPMENT OF GENECNN

The task of gene prediction has been largely stagnant in algorithmic improvements compared to when algorithms were first developed for predicting genes thirty years ago. Rather than iteratively improving the underlying algorithms in gene prediction tools by utilizing better performing models, most current approaches improve existing tools through incorporating increasing amounts of extrinsic data to improve gene prediction performance. The most common modeling algorithms in use for gene prediction are Hidden Markov Models (HMMs). Hidden Markov Models are problematic for gene prediction due to the Markovian assumption of independence. This assumption states that a given observation is solely dependent on the immediately prior state, but independent of all other states. This is not a viable assumption for genomes as both gene order and chromosomal location are often conserved, which is referred to as synteny (Renwick, 1971). This synteny among genes has been shown to exist for as long as 1.5 billion years in the case of genes found in the genomes of *Synechococcus* and *Arabidopsis* chloroplasts, which creates a large degree of spatial dependence among genes in evolutionarily related organisms (Zakharov, 2010). Given this genic spatial dependence, HMM-based gene prediction methods are flawed, which led to the development of a convolutional neural network (CNN) based approach named GeneCNN.

2.1 Convolutional neural networks

Convolutional neural networks are a type of neural network that are adept at generalizing spatial data. This spatial awareness allows convolutional neural networks to learn complex patterns and make predictions when faced with new data. The

approximation of data and classification of data points, in this case the classification of whether a sequence is genic or nongenic, is done by neural networks through nonlinear transformations (Goodfellow et al., 2016). These nonlinear functions often allow for neural networks to outperform other models, such as HMMs, by creating generalizations from complex datasets that linear functions alone cannot capture (Mohseni-Dargah et al., 2022).

As mentioned previously, CNNs are particularly useful when working with spatial data as they initially create generalizations of the input through scanning input data using a convolutional layer. Kernels, or matrices of a specific size, are used in these convolutional layers as sliding windows where a number of kernels with a specific depth known as a filter, are used to scan the entire input to summarize a given data point and its kernel-sized surroundings into a new data point. The output from these convolutional layers is used as input for pooling layers which summarize data with nonlinear functions, allowing for a dimensionality reduction while maintaining the defining aspects of the input data (Krizhevsky et al., 2017). This combination of convolutional layers and pooling layers allow for CNNs to learn complex patterns from spatial data, making them useful for *ab initio* gene prediction (Umarov & Solovyev, 2017).

A generalized version of GeneCNN relies first on an input sequence that is converted into a one-hot encoding format, changing the categorical representation of nucleotides into a numeric representation that can be used as input for GeneCNN (Figure 2-1). These one-hot encoded samples are then passed into a convolutional layer to generalize the sequences, the regularization technique batch normalization is applied, then the dimensionality reduction is performed by a pooling layer. For this project, this

combination of convolution, batch normalization, and pooling, was done three times in total. Once the data is sufficiently generalized, the output of the final pooling layer is passed to a flatten layer. This flatten layer converts the convolved output into a one-dimensional array which is able to be used as input for the dense layer. A dense layer, otherwise known as a fully connected layer, is a neural network implementation where neurons apply transformations to all input data points and are commonly used in CNNs after feature extraction with convolutional layers. Finally, output from this dense layer is passed through a softmax function which assigns probabilities for the given input to both the genic and nongenic classes.

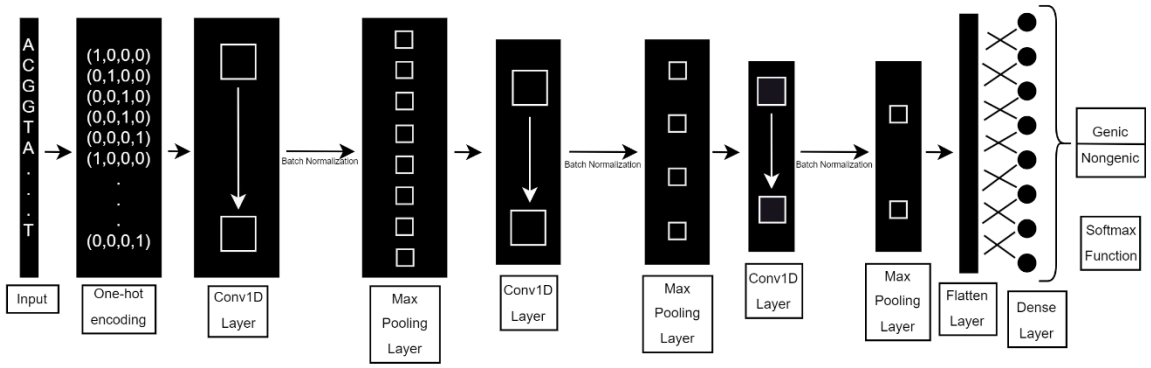

Figure 2-1. A generalized visual representation of GeneCNN. The input sequence is converted into a binary format through one-hot encoding. Convolutions are applied to the binary sequence through a Conv1D layer to add spatial information to each data point, batch normalization is used to standardize data before input into the max pooling layer which reduces data dimensionality by outputting only the max value per specified pool size. After repeating this combination of convolution, batch normalization, and max pooling twice more, results were input into a flatten layer to convert data into a one-dimensional format to input into a dense layer for linear transformations on all data points. The softmax function normalizes the output of the fully connected layer into a probability distribution for the two classes of genic and nongenic.

2.1.1 Data processing for CNNs

The overview of GeneCNN (Figure 2-1) does not capture the details of implementation. One of these implementation details is the common machine learning task of splitting input data into training, validation, and testing datasets. The training and validation datasets are both used during the model training process, where generalization of input occurs. The validation dataset exists to improve model performance during this training phase. The testing dataset is not shared with the model until after training is done to measure the performance of a model once input generalizations are defined for a given type of data. For a classification neural network such as this, predictions are made after the test data is input and performance is measured by comparing these predictions to the test dataset labels, which are the true values for the data, namely genic and nongenic for this project.

Balanced data is an important consideration for training a machine learning model. In a binary classification problem such as this CNN-based gene prediction approach, imbalanced data is the occurrence of a relatively large bias of data occurrences towards one class over the other. For binary data, a class ratio of 1.5:1 to 4:1 is considered a mildly imbalanced dataset, while a class ratio of 4:1 to 99:1 is considered a moderately imbalanced dataset (Google Developers, 2023). Imbalanced data becomes more of a performance detriment as the class imbalance gets larger due to the assumption made by machine learning algorithms that classes are balanced (Krawczyk, 2016). Due to this assumption of balance, a common result of training models on imbalanced data is that new data points will be predicted as the majority class, simply because this prediction was correct for the majority of the training period. One common data

processing approach used to regain class balance and improve model performance is data sampling. Data sampling can be done either by oversampling, where the minority class is randomly resampled by a complementary amount to the majority class, or undersampling, where the majority class is randomly sampled at an amount equal to the minority class rather than using the entire majority class of data.

2.1.2 Activation functions and optimization algorithms

Other major considerations to make when constructing a neural network include the activation function and optimization algorithms. Activation functions are functions designed to transform information into an input for the next layer (Sharma et al., 2020). Often these activation functions are used to create nonlinear transformations to allow a neural network to make robust generalizations of input data. The most commonly used activation function is currently the rectified linear unit (ReLU) (Bridle, 1990). ReLU is a popular activation function due to its simplicity, nonlinearity, and its general high performance across different types of models and datasets (Ramachandran et al., 2017).

Optimization algorithms are used in neural networks to iteratively change model parameters during the training period in accordance with how much a model deviates from the expected value in predictions. These algorithms are adjustable in the speed, or learning rate, at which they adjust model parameters to minimize model error. One commonly used optimization algorithm is Adaptive Movement Estimation (Adam) which was novel in its ability to dynamically update learning rates (Kingma & Ba, 2014). Kingma and Ba (2014) found that the Adam optimizer tends to outperform other similar optimization algorithms with its built-in bias correction. Due to this improvement and its

ease in implementation, Adam is often suggested for implementation in deep learning models as the best overall optimization algorithm (Ruder, 2017).

2.1.3 CNN overfitting and regularization

One general concern when working with any machine learning model is model overfitting. Overfitting occurs when a model creates poor data generalizations by instead creating generalizations that are too specific to the training data. This means that the model performance is artificially good for the training dataset, but poor for data that has not been encountered by the model before such as with the test dataset. To alleviate the occurrence of overfitting and reduce error, regularization methods are often applied. The two most common regularization methods are dropout and batch normalization.

Dropout is a regularization method that addresses model overfitting by randomly dropping entire neurons and their connections from a neural network during training (Srivastava et al., 2014). The proportion of these neurons that are randomly dropped during training can be adjusted to increase performance or reduce overfitting. When applying the neural network to the test dataset for assessing prediction performance, these dropped neurons are then reactivated with weights learned by the model to create better predictions with a lower error of data generalization. Srivastava et al. (2014) demonstrated that the implementation of dropout generally improved performance in neural networks but had the major drawback where inclusion of dropout leads to a doubling or tripling in computation time required to train a neural network, which is a substantial tradeoff to be considered.

Batch normalization is a regularization method that addresses the concern of model overfitting by performing normalization on the training data for each batch, which is the number of samples used for a single training period before the neural network performs a new iteration and updates weights (Ioffe & Szegedy, 2015). Batch normalization is advantageous as model performance can be improved with lower required computation time as well as reducing overfitting, potentially to the extent that implementing dropout is no longer necessary.

2.1.4 Hyperparameter optimization

The hyperparameters of a neural network are the set of all variables that determine how a model is trained, as well as its overall structure. These include the aforementioned number of convolutional layers, regularization techniques, activation functions, and learning rate. For convolutional neural networks, hyperparameters also include the exact size of kernels and filters used in convolutional layers. As there are many individual methods to tune, all of which can have a substantial impact on model performance, it is generally infeasible to achieve an optimally performing neural network through manual adjustment of hyperparameters alone. Instead, algorithms are used that iteratively adjust hyperparameters during a tuning process that selects for optimal performance of the neural network on the training and validation data by using different combinations of hyperparameter values. There are several different algorithms commonly used for tuning hyperparameters. The random search, Bayesian optimization, and Hyperband algorithms were all tested in this project.

Random search was among the first optimization algorithms to create a high throughput method of searching for optimal hyperparameters (Bergstra & Bengio, 2012).

The idea of this algorithm was to allow a random search space within a much larger range of values for each hyperparameter than typically done with grid search or manual methods. This increase in range allowed the random search algorithm to achieve similar or better results than previous methods in a lower amount of computation time.

Bayesian optimization is an optimization algorithm that initializes hyperparameters at random similar to random search but improves upon this randomness by searching for hyperparameter optima according to previous trial performance values (Močkus, 1974). Bayesian optimization has been shown to yield higher performing hyperparameters in a lower amount of computation time than a human expert using manual selection by 3% when tested on a sample dataset (Snoek et al., 2012).

Hyperband is the newest optimization algorithm of these three, which uses the random search algorithm as a foundation but speeds up the computation process through better resource allocation and early stopping of trials depending on how well the model with the selected hyperparameters is performing (Li et al., 2018). By improving the speed at which hyperparameter selection trials are computed, Hyperband indirectly improved model performance over random search by reaching optimal hyperparameters sooner. This performance improvement is particularly noticeable given a large hyperparameter search space, as native random search optimization becomes slow.

2.2 Measuring the performance of CNNs

There are various methods to evaluate the performance of a neural network through and after training. Loss, for example, is used during training to guide the model in creating fewer errors in its generalization of data. The measure of loss is an

optimization problem where the model distills the error between learned values and the expected values into a singular value with a specified loss function. To create a single value, a loss function is specified to be used by the model where it is often referred to as an objective function where the objective for the model is to reduce this loss value as much as possible given a set of hyperparameters. The loss function used in this project is binary cross-entropy, which measures the difference between predicted and expected values using a probability distribution created from the predicted values.

Accuracy, like loss, is another general performance metric used in machine learning. Accuracy as a performance metric is defined by the number of correct predictions divided by the total number of predictions made by the model. This simple percentage is popular for comparing models as a general overview of model performance but does not share the entire prediction set made by a model. Often a confusion matrix is used to visualize the number or percentage of classifications for any given type which gives a better indicator of performance than accuracy alone.

Classifications are distributed into four different types for a binary classification problem. True positive where the model correctly classified a sample as positive when the true value is positive. True negative where the model correctly classified a sample as negative when the true value is negative. False positive, otherwise known as type I error, where the model incorrectly classified a sample as positive when the true value is negative. False negative, or type II error, when the model incorrectly classified a sample as negative when the true value is positive. These different classification sets show the full performance of a given model by sharing not only when the model incorrectly classifies a sample, but how it misclassified a sample relative to the true value.

Another commonly used performance metric is the F_1 score, which is a measure of how well the model classified positive and negative samples relative to each other. To calculate the F_1 score, precision and recall are used. Precision is the set of true positive results divided by the total set of positive results including misclassifications. Recall is the set of positive results divided by the number of misclassified negative samples, which can also be computed by taking the complement of the type II error rate. The harmonic mean of the precision and recall is the F_1 score.

2.3 Developing GeneCNN for gene prediction

The development of GeneCNN was designed for gene prediction using buffalograss as a model system. In collaboration with Dr. David Huff, we sequenced the buffalograss genome. With the newly sequenced buffalograss genome, the overarching project goal was to create a gene prediction method that does not rely on HMMs and instead utilize a CNN. A supervised learning approach was used for GeneCNN which is the use of known output data to train the neural network to classify unknown data into the same individual classes used in the known output data for training. To create a supervised CNN for gene prediction, labels, or true values for each nucleotide in the genome were generated using transcriptomic data found in the BioProject database provided by the National Center for Biotechnology Information (NCBI). This database provides multiple types of information gathered about an organism for a given project.

A major consideration when creating GeneCNN is the minimization of the false positive rate, or type I error. As gene prediction is often one of the first major steps in a workflow dedicated to understanding the genome of an organism, gene prediction models

must not be prone to polluting downstream aspects of the workflow with sequences that were predicted to be genic when these sequences are truly nongenic.

Methods

2.4 Preparation of buffalograss genome by Dovetail Genomics

Genomic DNA was extracted from a male diploid buffalograss [*Bouteloua dactyloides* (Nutt.) Columbus] genotype PI 578219 that originated from central Mexico and used for subsequent experimentation (Wu and Lin, 1994). The genomic DNA was sent to Dovetail Genomics for sequencing and *de novo* genome assembly. Dovetail Genomics performed genome sequencing using the PacBio HiFi platform (Wenger et al., 2019). Scaffolding was done using Omni-C proximity ligation with HiRise scaffolding software (Putnam et al, 2016). Genomic data was returned from Dovetail Genomics as a FASTA file with 2,498 scaffolds, the first 10 of which were described as pseudochromosomes due to their length. The 2,498 scaffolds were globally aligned against the pseudo-chromosomes with BBMap to verify completion of the genome (Marić, 2015).

2.5 Initial annotation with BRAKER2

A traditional genome annotation approach was taken using BRAKER2 on the unmodified genome file provided by Dovetail Genomics. This process started with RepeatModeler2 to identify transposable elements and long terminal repeats (Flynn et al., 2020). RepeatModeler2 was first used to build a database on the genome using RMBlast, a modified version of NCBI blastn. Using the newly built genome-specific database, RepeatModeler2 was run separately, either with long terminal repeats classified or

without. Repeats identified in the genome from the model libraries were then soft masked with RepeatMasker (Flynn et al., 2020; Smit et al., 2013-2015).

Transcriptomic data that was provided to Dovetail Genomics include RNAs found in BioProjects PRJNA237791, PRJNA297834, PRJNA578934, and PRJNA231727. Bioproject PRJNA237791 was used for transcriptomic datasets SRR1166811 and SRR1166718, which contained 3.4 and 3.8 billion nucleotides respectively. Bioproject PRJNA297834 was used for the transcriptomic dataset SRR3188108, which contained 2.7 billion nucleotides. Bioproject PRJNA578934 was used for transcriptomic datasets SRR10326369 and SRR10326347, which contained 2.9 and 2.8 billion nucleotides respectively. Bioproject PRJNA231727 was used for the transcriptomic dataset SRR1051213, which contained 5.6 billion nucleotides. These datasets were used for subsequent gene prediction approaches as genic evidence. RNAs were trimmed using Trim Galore to remove sequence adapters and low-quality reads (Krueger, 2019). Cleaned reads were mapped against the buffalograss genome using BBMap with default parameters (Marić, 2015). Mapped reads were converted to binary format and sorted using Samtools (Li et al., 2009). Gene prediction and annotation were done with BRAKER2 using the soft masked buffalograss genome and the mapped RNA reads as evidence (Brůna et al., 2021). Initial prediction led to more genes predicted than anticipated, therefore the genome was reduced by removing contaminants and repeat elements from the additional scaffolds.

2.6 Removal of contaminants and repeat elements from the buffalograss genome

The 2,498 additional scaffolds were compared against the 10 pseudochromosomes with CD-HIT-2D, where scaffolds that exceeded 70% identity of scaffold length to

pseudochromosomes were removed (Fu et al., 2012). The remaining scaffolds were subject to BLAST, specifically discontiguous MegaBLAST, to identify sequences taxonomically and allow for removal of potential contaminants, namely non-plant related sequences (Altschul et al., 1990; Camacho et al., 2009). The pseudochromosomes were mapped against the remaining scaffolds with BBMap for discovering scaffold coverage relative to the pseudochromosomes. BLAST results, the mapping results, and remaining scaffolds were input to BlobToolKit to determine scaffolds to remove based on taxonomy associated to a different phylum, coverage, and GC content (Challis et al., 2019). All scaffolds that had BLAST hits to any phylum other than no-hit or Streptophyta were removed. Remaining scaffolds were then subject to removal based on coverage to remove redundancy, where all scaffolds that had a coverage of greater than 2 were removed. Scaffolds that had a coverage of less than 1 were checked by mapping against transcriptomic data from the aforementioned BioProjects with BBMap to ensure there were no missing genic elements from the pseudochromosomes, but all scaffolds with a coverage of less than 1 were ultimately discarded due to low mapping rates of under 4% of these low coverage scaffolds mapping to the transcriptomic data. Lastly, scaffolds that had GC content lower than 30.6% or higher than 60% were removed, producing the newly reduced genome.

Using the newly reduced genome, a workflow was developed for utilizing a CNN-based approach for gene prediction, GeneCNN, rather than traditional HMM-based gene prediction approaches used in tools such as BRAKER2. To begin data preparation input for GeneCNN, the cleaned genome was first flattened to allow for use of all pseudochromosomes and cleaned scaffolds. Flattening of the genome was done by

reducing the 308 sequence entries, the 10 pseudochromosomes and 298 additional scaffolds, into a singular sequence in rank size order. A check was performed by mapping the reduced genome to this flattened genome to ensure there were no chimeric genes formed through the connection of scaffolds in rank size order. This newly refined genome was used for subsequent analysis which included labeling with transcriptomic data and classification with GeneCNN.

2.7 GeneCNN development

The genome was labeled for supervised classification which used transcriptomic data from the previously used four BioProjects as a foundation. Samtools with the option ‘depth -aa’ was used to generate mapping depth across all transcriptomic data per nucleotide of the genome. This result was aggregated for use in creating a binary set of labels by taking a sum of mapped nucleotides per nucleotide position in genomic data. If the sum of depth was nonzero at a given position, it was given a label of 1 where this label represents a genic nucleotide. Conversely, if the mapping depth at the given nucleotide was zero, then the nucleotide position was given a label of 0 which represents a nongenic nucleotide.

Due to the genome being flattened, using each pseudochromosome or additional scaffold as an individual sample for input to GeneCNN was not an option. Instead, samples were created by taking all contiguously labeled sequences, both genic and nongenic, of a specific length. Genic labeled samples were representative of exons, while nongenic labeled samples were representative of noncoding sequences. The original length for these samples was 200 nucleotides but has since been updated to 500 nucleotides for improved performance at a cost of creating a larger imbalance between

the number of genic and nongenic samples. This iterative process of testing GeneCNN performance with different sample lengths was done with values of 200 nucleotides to 800 nucleotides in increments of 100 nucleotides, where performance increased until 500 nucleotide length samples but stagnated or worsened after.

These longer contiguously labeled sequences lowered the nongenic to genic class ratio from its original 3.1:1 to 4.9:1. To combat this problem and improve classification performance, undersampling the nongenic class and oversampling the genic class were tested. Ultimately, oversampling resulted in higher classification performance and therefore was used for subsequent analysis. Oversampling was done through the use of the entire genic class with the addition of random sampling of the genic class in a complementary amount to the nongenic class, making the new class ratio a balanced 1:1.

As the two classes were held in separate data structures, they were now merged into the same data structure where the labels were modified from a per nucleotide format to a per sample format. To prepare the samples for conversion to one-hot encoding for binary classification, the alphabetic representation of nucleotide bases was converted into a numeric representation. This alphabetic representation of nucleotides was changed into a one-hot encoding format as follows: ‘A’ to (1, 0, 0, 0), ‘C’ to (0, 1, 0, 0), ‘G’ to (0, 0, 1, 0), and ‘T’ to (0, 0, 0, 1).

The data was split using scikit-learn into training, validation, and testing datasets at a ratio of 75%, 15%, and 10% respectively (Pedregosa et al., 2011). After the data split, each dataset was converted to a one-hot encoding format using TensorFlow (Abadi et al., 2016). These split datasets were subsequently used in a CNN built using TensorFlow. GeneCNN is composed of three Conv1D layers, each of which use the

ReLU activation function to introduce non-linearity in training (Fukushima, 1969). Filter and kernel sizes for these Conv1D layers were determined by RandomSearch from KerasTuner (O'Malley et al., 2019). Each of these Conv1D layers were followed by a MaxPooling1D layer to reduce dimensionality. These MaxPooling1D layers used a pool size of 4, which only outputs the maximum value found per 4 data points. Following this triplicate of Conv1D and MaxPooling1D layers are the fully connected layer with ReLU activation and the output softmax layer which outputs probabilities for the given samples for both classes, genic and nongenic. GeneCNN uses a binary cross-entropy loss function and Adam optimizer (Shannon, 1948).

2.8 GeneCNN performance improvements

To make gene predictions with GeneCNN in a manner similar to other gene prediction methods, repeat masking was performed on the genome. Repeat masking was done using the same method for the original BRAKER2 approach by using RepeatModeler2 followed by RepeatMasker. This change in masking the genome improved loss and accuracy but reduced predictive power. As it did not have a major impact on GeneCNN performance, the masked genome was used as the final dataset for all results unless otherwise specified.

The initial iteration of GeneCNN was composed of a single Conv1D layer but was updated to three layers in sequence as this update improved classification accuracy. Each of these Conv1D layers were originally parameterized by randomly chosen values using RandomSearch from KerasTuner with manual adjustment afterwards. This hyperparameter optimization has since been updated to using BayesianOptimization from KerasTuner. The Hyberband optimization method from KerasTuner was also attempted

but achieved marginally worse results than BayesianOptimization. Using BayesianOptimization for tuning GeneCNN parameters increased performance but led to a large increase in model overfitting, so batch normalization was introduced as a regularization method to reduce this model overfitting.

Results

2.9 Initial annotation with BRAKER2

As an exploratory approach with a prominent existing traditional gene prediction tool, BRAKER2 was used on the original genome file provided by Dovetail Genomics. The number of coding genes predicted by BRAKER2 was 40,169. When only the 10 pseudochromosomes were used for gene prediction, the total predicted gene count was 31,941. As the only difference between the two approaches was the inclusion of the additional scaffolds, these scaffolds were investigated for potential genic content. The investigation discovered that there was a large degree of redundancy in the additional scaffolds relative to the 10 pseudochromosomes, as well as a low level of contamination. This information led to the development of a reduced genome file, where an approach was taken that rather than removing all other scaffolds completely, scaffolds that contained potential genic sequences not found in the pseudochromosomes were retained; these retained additional scaffolds had relatively low levels of redundant information compared with the 10 pseudochromosomes.

2.10 Removal of contaminants and repeat elements from the buffalograss genome

The initial removal of 70% sequence identity of scaffold length to pseudochromosomes removed 518 redundant scaffolds out of the 2,488 non-

pseudochromosome scaffolds provided by Dovetail Genomics. Contamination was also a concern, and 10 scaffolds were removed on the basis of taxa, where the scaffolds were removed if they were assigned a taxon by BLAST of anything other than Streptophyta or no-hit. Removal of outliers on the basis of GC content led to the reduction in the scaffold dataset by 15 scaffolds. Based on redundancy through global alignment, 1,647 scaffolds were removed, leaving the final dataset with 10 pseudochromosomes and 298 additional scaffolds. These removals were done on the basis of coverage of the pseudochromosomes onto the scaffolds, where scaffolds that had a coverage value of 1 to 2 were retained and all other scaffolds were removed. Scaffolds with a coverage value of 2 or greater were deemed redundant with the pseudochromosomes, and low coverage scaffolds were shown to not have much sequence similarity with the pseudochromosomes due to a 4% global alignment rate. This newly modified genome was then flattened for GeneCNN development. The flattened genome was checked against the genome with BBMap where the difference between the two was in the range of 0.0001 to 0.005 in mapping percentage, showing that there was likely no loss in biological information from flattening the genome.

2.11 GeneCNN development

Labeling of every nucleotide in the genome was done using Samtools with option ‘depth -aa’, which resulted in 269,460,553 nucleotides being labeled nongenic, and 86,962,341 nucleotides being labeled as genic, making for a class ratio of 3.1:1 respectively. To test the effect of sequence length on the performance of GeneCNN, sample sequences were tested from 200 nucleotide length to 800 nucleotide length in increments of 100 nucleotides, where the 500 nucleotide length sequences resulted in the

highest performance for GeneCNN, so these 500 nucleotide length sequences were used as samples for GeneCNN going forward. The number of labeled sequences at 500 nucleotide lengths were 475,509 nongenic sequences and 97,164 genic sequences. This use of longer sample sequences to increase GeneCNN performance did have an adverse effect on the balance of the two classes, where the new class ratio was 4.9:1. While the 500 nucleotide length sequences had the best performance of sequence lengths tested, this relatively large class imbalance led to poor predictive performance by GeneCNN long-term and so oversampling of the genic class was performed to regain class balance. The entire 97,164 genic labeled sequences were used along with a complementary random sampling of 378,345 sequences from the set of genic sequences with replacement, making the class ratio now balanced 1:1.

GeneCNN parameterization was defined for the majority of the project by RandomSearch from KerasTuner, with manual tuning of all hyperparameters other than the learning rate to improve both performance through the reduction in overfitting as well as improve classification accuracy. Replacing RandomSearch with BayesianOptimization from KerasTuner improved performance but largely increased the model overfitting which led to batch normalization being used. Batch normalization was implemented from Keras after every convolutional layer which led to a reduction in overfitting and a boost to GeneCNN performance on the genome. The final hyperparameter values, including those selected by the BayesianOptimization tuner trials, are described in Table 2-1. The impact of these final hyperparameter values on the input and output sizes of GeneCNN on a per-layer basis of the model is shown in Figure 2-2.

Table 2-1. Final GeneCNN hyperparameters for the first major version of the gene prediction tool. The hyperparameters chosen by BayesianOptimization of Keras Tuner include all kernel sizes, filters, fully connected layer units, and Adam optimizer learning rate. No suffix indicates the first layer used in sequential order of the model, while the suffixes '_1' and '_2' indicate the second and third layers used respectively.

Layer	Hyperparameter values
Conv1D	Kernel size = 3, filters = 116
BatchNormalization	—
MaxPooling1D	Pool size = 4
Conv1D_1	Kernel size = 5, filters = 120
BatchNormalization_1	—
MaxPooling1D_1	Pool size = 4
Conv1D_2	Kernel size = 5, filters = 80
BatchNormalization_2	—
MaxPooling1D_2	Pool size = 4
Fully Connected Layer	Units = 124
Softmax	Units = 2
Adam Optimizer	Learning rate = 0.001

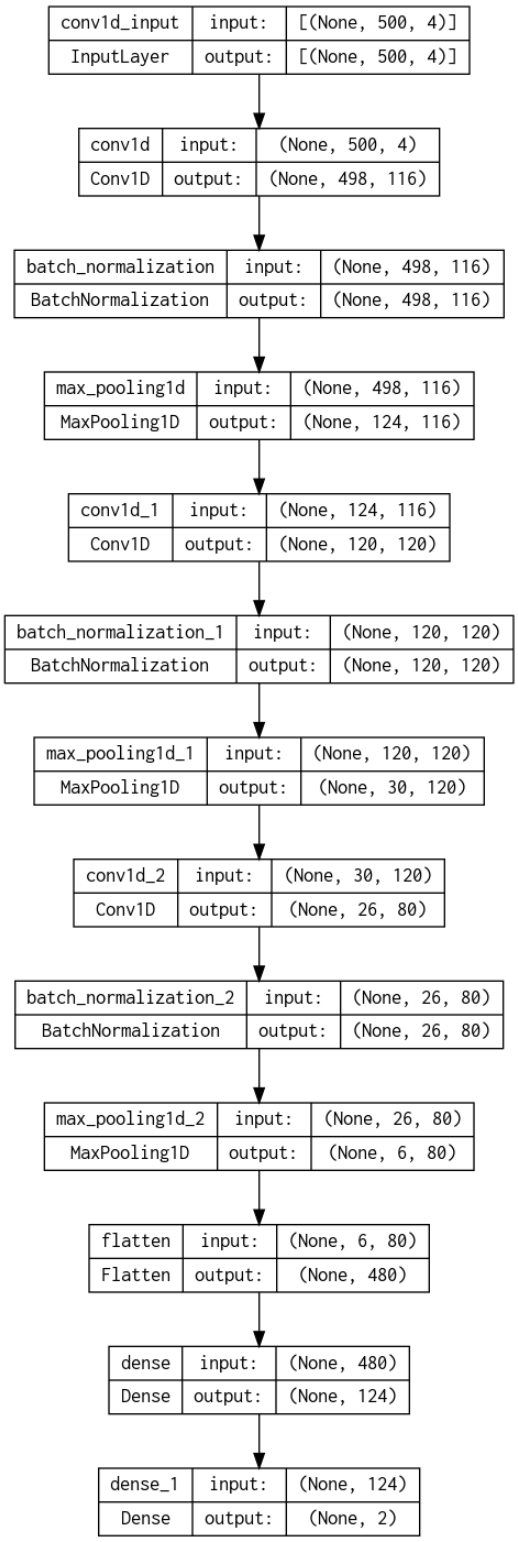

Figure 2-2. GeneCNN layout in sequential order of layers and operations as output from Tensorflow plot_model. All layer input and output sizes are shown with numeric values, where a singular value represents a one-dimensional array, and two values represent a

two-dimensional matrix. The ‘None’ values found in all input and output sizes are in the tuple location to define batch size, but due to TensorFlow utilizing the default batch size of 32, ‘None’ is shown as a value instead. No suffix found in the top left cell per table indicates the first layer used in the sequential order of the model, while the suffixes ‘_1’ and ‘_2’ indicate the second and third layers used respectively. Conv1D layers apply 1-dimensional convolutions, batch normalization layers apply regularization to the data, max pooling layers perform dimensionality reduction, and the flatten layer transforms the data into a 1-dimensional array to deliver input to the first dense layer which is a fully connected neural layer. The second dense layer applies the softmax function, assigning a probability to genic and nongenic class predictions.

2.12 GeneCNN performance

GeneCNN has undergone several iterations to improve performance, all of the major iterations are listed in Table 2-2. The major iterations were all either a major branching point in the data preprocessing or led to model changes that improved GeneCNN performance. Rather than using historical GeneCNN results which differ completely in hyperparameters and training length, all major model iterations were recreated where they differ from the current model in only the manner listed to allow for best case comparisons between iterations. Results from historical values where the model types shown in Table 2-2 were initially ran can be found in Supplemental Figures A-1 through A-15. The metric ‘Overfit’ was created for Table 2-2 to give a tabular representation of loss and accuracy between the training and validation datasets but is not a replacement for a representation of change over time.

The model types for Table 2-2 consist of the ‘200 nt’ model which uses 200 nucleotide (nt) length sample sequences rather than the current 500 nucleotide length samples, ‘1 Conv Layer’ model which only uses a singular convolutional layer as opposed to the current three convolutional layers, ‘4.9:1 Class Ratio’ model that forgoes

the use of random oversampling where it instead uses the imbalanced genome as is, 'Manual Adjust' model that uses RandomSearch from KerasTuner with the final parameter values decided by manual adjustment, 'No BN' model which is a removal of the batch normalization regularization from the current CNN model, and 'Current' which is the current iteration of GeneCNN.

Table 2-2. GeneCNN performance across different methods during development. Yellow highlighted cells are the highest performing for a given column. Overfit is defined as |training – validation| where the first overfit column is relevant for loss, and the second overfit column is relevant for accuracy. The confusion matrix section of Table 2-2 contains the columns 'TP' true positive, 'TN' true negative, 'FP' false positive, and 'FN' false negative. These columns are the prediction accuracy on a specific class using the test data.

	Loss			Overfit	Accuracy			Overfit	Confusion matrix				F_1 score
Model	Training	Validation	Test		Training	Validation	Test		TP	TN	FP	FN	
200 nt	0.412	0.461	0.463	0.049	0.808	0.784	0.783	0.024	0.89	0.68	0.11	0.32	0.783
1 Conv Layer	0.274	0.35	0.353	0.076	0.887	0.862	0.861	0.025	0.94	0.79	0.06	0.21	0.861
4.9:1 Class Ratio	0.119	0.459	0.458	0.34	0.95	0.867	0.867	0.083	0.53	0.94	0.47	0.06	0.867
Manual Adjust	0.391	0.395	0.392	0.004	0.825	0.825	0.826	0	0.88	0.77	0.12	0.23	0.826
No BN	0.186	0.303	0.309	0.117	0.93	0.895	0.896	0.035	0.93	0.86	0.07	0.14	0.896
Current	0.078	0.191	0.184	0.113	0.972	0.946	0.948	0.026	0.97	0.92	0.03	0.08	0.948

The current method outperforms nearly all other methods tested except for in overfit where the RandomSearch tuning combined with manual adjustment outperformed it, and the negative class predictions where it was outperformed by the imbalanced genome. The 'Manual Adjust' model greatly outperformed all other methods when it came to the lowest degree of overfit, however this performance did not translate to a higher classification performance than the current method as shown by the confusion

matrix and F_1 score values. The use of imbalanced data found in the '4.9:1 Class Ratio' model led to a 2% improvement in classification on the negative class compared to the current method as shown in the TN and FN columns, but much worse performance in classification on the positive class, where this iteration of the model only achieved accurate genic class predictions on 53% of the test data (Table 2-2).

Model overfitting is easier to notice in graphical format than in a table, as the trends in training and validation datasets can be compared over the training period measured in epochs, which are individual passes of the training data through the model. Overfitting as a measure of model loss over the course of the training period is evident in Figure 2-3, where the validation loss begins increasing over the training period while the training loss continues to decrease. This beginning of an inverse relationship between validation loss and training loss affirms that the generalizations made by the model using the training data begin to become poor for withheld data and is why the model was stopped from further training at 20 epochs. Meanwhile, overfitting as a measure of model accuracy was at a lower level for the current method (Table 2-2). To verify that the level of overfitting for accuracy is low and not just due to variance in the final epoch, a graph is used (Figure 2-4). This graph verifies that the level of model overfitting is low when measured via accuracy, as the validation accuracy shows high performance relative to the training accuracy, with only a minor level of variance observed once the validation accuracy plateaus at approximately 94.5%.

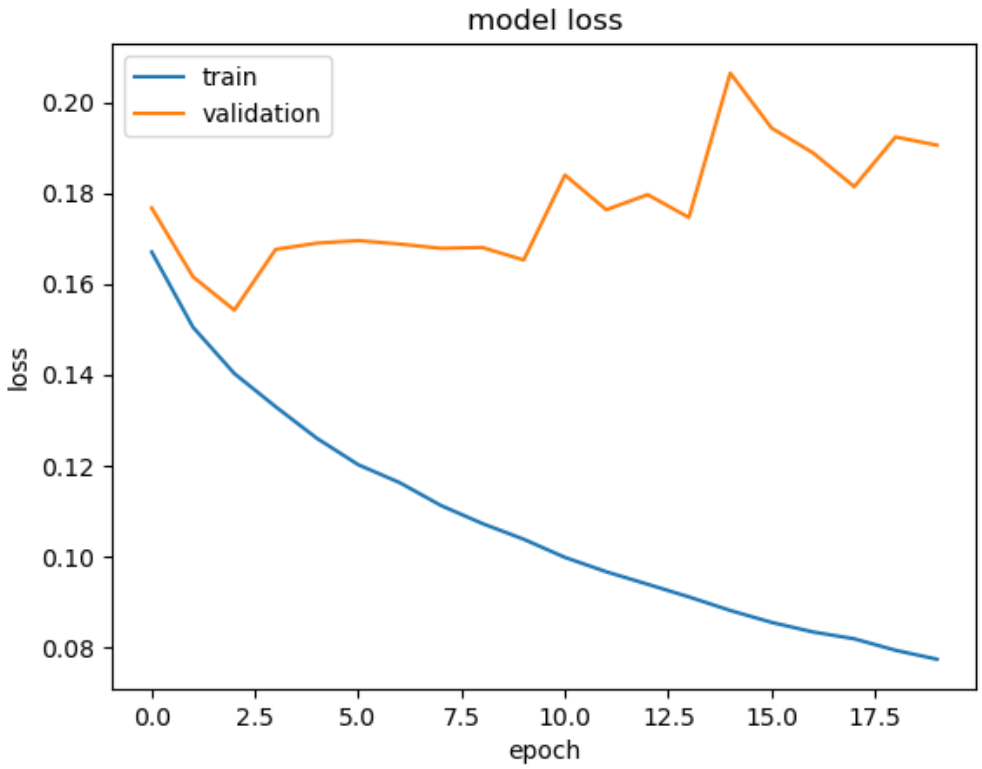

Figure 2-3. A graphical display of the model loss over time using the current iteration of GeneCNN. Loss is shown over a training period of 20 epochs with both training and validation datasets. Model overfitting can be seen through the increasing degree of loss in the validation data relative to the training data during the training period.

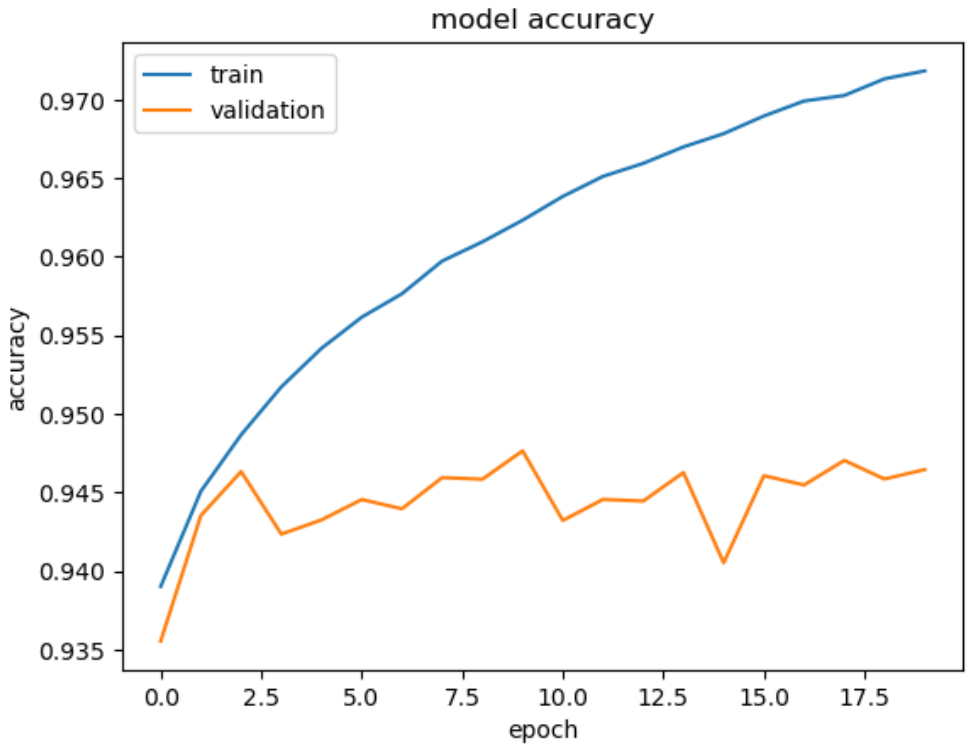

Figure 2-4. A graphical display of the model accuracy over time using the current iteration of GeneCNN. Accuracy is shown over a training period of 20 epochs with both training and validation datasets. Model overfitting is not evident here but there is a minor level of variance in the validation accuracy once it plateaus at approximately 94.5%.

Discussion

Several different paths were explored for developing GeneCNN for buffalograss gene prediction and its iterative improvements (Table 2-2). The objective of improving GeneCNN was to minimize the type I error (the 'FP' column of Table 2-2), as this minimizes the number of genes falsely predicted. Since gene prediction is a major step in workflows such as gene annotation, this minimization of falsely predicted genes is important to not adversely affect downstream analysis. Minimization of type I error typically came at a cost to the overall accuracy and loss, especially through overfitting.

This is seen in the current iteration where GeneCNN has relatively high overfitting compared to the other model iterations, especially in loss (Table 2-2). Of the different model iterations tested, the current model iteration performs the best overall. The current GeneCNN iteration reduced overfitting by a small margin with the introduction of batch normalization yet maintains a low type I error rate of 3%, which keeps genic classification largely pure for downstream analysis.

Of the iterations tested for both GeneCNN accuracy and hyperparameter optimization, the current set of hyperparameters yield the best overall performance. Though the BayesianOptimization tuning method iterates relatively slowly for tuning hyperparameters compared to random search or Hyperband, its parameter selection is much more informed due to its hyperparameter search being dictated by the performance of the previous iterations rather than random selection. Given that the BayesianOptimization method reached higher performance for GeneCNN in the same number of computational cycles as Hyperband, which is known for its fast convergence on optimal hyperparameters, there is confidence in the parameters provided by the BayesianOptimization tuning method given the number of trials used.

The current method for GeneCNN was outperformed in the negative class predictions shown in the ‘4.9:1 Class Ratio’ model in Table 2-2. This result of better negative class prediction was due to the large bias of negative class samples provided to GeneCNN for learning. Given this large class imbalance, the model learning is biased towards the negative class to the extent that it begins to make negative predictions solely because negative predictions are correct the majority of the time, rather than learning generalizations for the sample data. This bias in learning due to sample class imbalance is

a well-known problem in machine learning and is why random oversampling was performed to correct for this issue.

Model overfitting is a common problem in deep learning and there are two common regularization approaches used to alleviate this which include dropout and batch normalization. Batch normalization alone was used for the current implementation of GeneCNN as it has been shown that dropout generally results in poor performance when used in a CNN (Garbin et al., 2020). In the study done by Garbin et al. (2020), it was demonstrated that there are particular cases where dropout may improve model performance in CNNs, but the implementation and experimentation for dropout takes a large amount of time. Additionally, not every model case has this potential to be improved by dropout.

To further improve the performance of GeneCNN, dropout should be explored at multiple rates of neuron dropout to potentially identify a method for reducing the amount of model overfitting, assuming that GeneCNN has the potential for improvement via dropout. This rate of model overfitting, particularly for measuring loss, is currently the worst performing aspect of GeneCNN so the ideal solution would be to find a rate of dropout that reduces overfitting without worsening prediction accuracy. Additionally in testing various rates of dropout, there is potential for further improving hyperparameter optimization through more trials with BayesianOptimization. Improvements to hyperparameter optimization could result in lower overfitting or improved classification accuracy. No other major potential model improvements for training performance are expected beyond the aforementioned dropout and hyperparameter optimization.

GeneCNN has proven that genic sequences can be accurately predicted with a convolutional neural network-based approach, where genic sequence predictions yielded a 97% accuracy rate. However, this performance was only measured on processed data, where the test dataset has a large degree of similarity to the training dataset. To further evaluate the performance of GeneCNN, unprocessed buffalograss pseudochromosome sequences must be used for gene prediction. This performance evaluation of GeneCNN includes validation with external data as well as comparing the performance of GeneCNN with other gene prediction tools.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2016). TensorFlow: Large-Scale machine learning on heterogeneous distributed systems. *ArXiv Preprint ArXiv:1603.04467*. http://arxiv.org/abs/1603.04467

Altschul, S., Gish, W., Miller, W., Myers, E., & Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology*. *215*(3), 403–410.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, 281–305. https://doi.org/10.1016/S0022-2836(05)80360-2

Brůna, T., Hoff, K. J., Lomsadze, A., Stanke, M., & Borodovsky, M. (2021). BRAKER2: Automatic eukaryotic genome annotation with GeneMark-EP+ and AUGUSTUS supported by a protein database. *NAR Genomics and Bioinformatics*, *3*(1). https://doi.org/10.1093/nargab/lqaa108

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *NATO ASI Series (Series F: Computer and Systems Sciences)*, *68*. https://doi.org/10.1007/978-3-642-76153-9_28

Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., & Madden, T. L. (2009). BLAST+: Architecture and applications. *BMC Bioinformatics*, *10*, 1–9. https://doi.org/10.1186/1471-2105-10-421

Challis, R., Richards, E., Rajan, J., Cochrane, G., & Blaxter, M. (2019). BlobToolKit – Interactive quality assessment of genome assemblies. *G3: Genes, Genomes, Genetics*, *10*(4), 1361-1374. https://doi.org/10.1534/g3.119.400908

Flynn, J., Hubley, R., Goubert, C., Rosen, J., Clark, A., Feschotte, C., & Smit, A. (2020). RepeatModeler2 for automated genomic discovery of transposable element families. *Proceedings of the National Academy of Sciences*, *117*(17), 9451–9457. https://doi.org/10.1186/s13059-018-1577-z

Fu, L., Niu, B., Zhu, Z., Wu, S., & Li, W. (2012). CD-HIT: Accelerated for clustering the next-generation sequencing data. *Bioinformatics*, *28*(23), 3150–3152. https://doi.org/10.1093/bioinformatics/bts565

Fukushima, K. (1969). Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, *5*(4), 322–333. https://doi.org/https://doi.org/10.1109/TSSC.1969.300225

Garbin, C., Zhu, X., & Marques, O. (2020). Dropout vs. batch normalization: An empirical study of their impact to deep learning. *Multimedia Tools and Applications*, *79*(19–20), 12777–12815. https://doi.org/10.1007/s11042-019-08453-9

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Google Developers. (2023). https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 448–456. http://arxiv.org/abs/1502.03167

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference for Learning Representations*. http://arxiv.org/abs/1412.6980

Krawczyk, B. (2016). Learning from imbalanced data: Open challenges and future directions. In *Progress in Artificial Intelligence, 5*(4), 221-232. https://doi.org/10.1007/s13748-016-0094-0

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Krueger, F. (2019). *Trim Galore* (0.6.5).

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, *25*(16). https://doi.org/10.1093/bioinformatics/btp352

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, *18*, 1–52. http://arxiv.org/abs/1603.06560

Marić, J. (2015). Long Read RNA-Seq Mapper. *University of Zagreb-Faculty of Electrical Engineering and Computing-Master Thesis No*, *1005*.

Močkus, J. (1974). On Bayesian methods for seeking the extremum. *Optimization Techniques IFIP Technical Conference: Novosibirsk*, 400–404.

Mohseni-Dargah, M., Falahati, Z., Dabirmanesh, B., Nasrollahi, P., & Khajeh, K. (2022). Machine learning in surface plasmon resonance for environmental monitoring. *Artificial Intelligence and Data Science in Environmental Sensing*, 269–298. https://doi.org/10.1016/B978-0-323-90508-4.00012-5

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., & Invernizzi, L. (2019). *KerasTuner*. https://github.com/keras-team/keras-tuner

Pedregosa, F., Michel, V., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Vanderplas, J., Cournapeau, D., Pedregosa, F., Varoquaux, G., Gramfort, A., Thirion, B., Grisel, O., Dubourg, V., Passos, A., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. http://scikit-learn.sourceforge.net.

Putnam, N. H., O'Connell, B. L., Stites, J. C., Rice, B. J., Blanchette, M., Calef, R., Troll, C. J., Fields, A., Hartley, P. D., Sugnet, C. W., Haussler, D., Rokhsar, D. S., & Green, R.

E. (2016). Chromosome-scale shotgun assembly using an in vitro method for long-range linkage. *Genome Research*, *26*(3), 342–350. https://doi.org/10.1101/gr.193474.115

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. *ArXiv Preprint ArXiv:1710.05941*. http://arxiv.org/abs/1710.05941

Renwick, J. H. (1971). The mapping of human chromosomes. *Annual Review of Genetics*, *5*(1), 81–120. https://doi.org/10.1146/annurev.ge.05.120171.000501

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *ArXiv Preprint ArXiv:1609.04747*. http://arxiv.org/abs/1609.04747

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379–423.

Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, *4*(12), 310–316. http://www.ijeast.com

Smit, AFA, Hubley, R., & Green, P. (2013-2015). *Repeat Masker Open-4.0* (4.0).

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, *25*.

Srivastava, N., Hinton, G., Krizhevsky, A., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Umarov, R. K., & Solovyev, V. v. (2017). Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. *PLoS ONE*, *12*(2). https://doi.org/10.1371/journal.pone.0171410

Wenger, A. M., Peluso, P., Rowell, W. J., Chang, P. C., Hall, R. J., Concepcion, G. T., Ebler, J., Fungtammasan, A., Kolesnikov, A., Olson, N. D., Töpfer, A., Alonge, M., Mahmoud, M., Qian, Y., Chin, C. S., Phillippy, A. M., Schatz, M. C., Myers, G., DePristo, M. A., … Hunkapiller, M. W. (2019). Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*, *37*(10), 1155–1162. https://doi.org/10.1038/s41587-019-0217-9

Wu, L., & Lin, H. (1994). Identifying buffalograss [*Buchloe dactyloides* (Nutt.) Engelm.] cultivar breeding lines using random amplified polymorphic DNA (RAPD) markers. *Journal of the American Society for Horticultural Science, 119*(1), 126-130. https://doi.org/10.21273/JASHS.119.1.126

Zakharov, I. A. (2010). Location of genes in chromosomes: Random or not? *Russian Journal of Genetics*, *46*(10), 1165–1172. https://doi.org/10.1134/S1022795410100066

CHAPTER 3: ASSESSMENT AND VALIDATION OF GENECNN

The creation of algorithms for predicting genes in sequenced genomic data is a field of study that has spanned thirty years. Many types of algorithms have been developed for gene prediction, the most prevalent of which are based on Hidden Markov Models (HMMs). Hidden Markov Models have proven to be commonly used in many fields, including computational biology, due to their relative ease of implementation and computational simplicity compared to other statistical models. However, when used as a statistical model in gene prediction, HMMs are built upon a faulty statistical assumption, namely the Markovian assumption of independence. This assumption states that any given observation is entirely dependent on the immediately prior state within the path of states that comprise a HMM, while independent of all other states (Henderson, 1997). This assumption is certainly flawed in the case of gene prediction, as an example, syntenic genes are known to maintain both their relative chromosomal ordering and location in evolutionarily related organisms, even in large evolutionary distances (Renwick, 1971; Zakharov, 2010). A new gene prediction tool, GeneCNN, was developed and avoids reliance on HMM-based gene prediction models. GeneCNN is a gene prediction tool that uses convolutional neural networks (CNNs) for modeling the structure of genic sequences. Convolutional neural networks do not assume that data is independent like HMMs, rather they use spatial dependence to their advantage by creating new data where datapoints are transformed using their surrounding datapoints. GeneCNN achieved 97% accuracy using test data for correctly predicting genic sequences when trained on buffalograss [*Bouteloua dactyloides* (Nutt.) Columbus] (Chapter 2). However, to fully evaluate the performance of GeneCNN as a gene

prediction tool, its performance must be compared against existing gene prediction tools and validated with known gene sequences.

3.1 Predicting genes with GeneCNN

To develop GeneCNN, buffalograss was used as a model system. GeneCNN was trained using a heavily modified version of the buffalograss genome, where samples were created using contiguously labeled nucleotides defined by transcriptome alignment. GeneCNN was then used for predicting genes using a shortened but otherwise unmodified sequenced genome of buffalograss. GeneCNN is a supervised CNN, meaning labeled data is used to train the model to predict the class of given input data, in this case whether the input sequence is genic or nongenic. Labels are the assignment of a distinct class to a data point, or in the case of GeneCNN, the assignment of a singular nucleotide as genic or nongenic. These label assignments were established across the buffalograss genome using publicly available transcriptomic data from BioProjects PRJNA237791, PRJNA297834, PRJNA578934, and PRJNA231727. Training data for GeneCNN was created through collecting contiguous sequences of a single label type, which represented exons and noncoding sequences for positive and negative labels respectively. Iterative improvements to GeneCNN established a high performing CNN model, allowing the tool to then be used for predicting genes given genomic sequences.

3.1.1 Gene prediction workflow

The trained GeneCNN was used for gene prediction where 500 nucleotide length sequences are provided as input. For optimal performance, the number of sequences should be a multiple of the batch size provided to TensorFlow. The batch size used by

GeneCNN is 32, the default for TensorFlow. Once input sequences are provided to GeneCNN in prediction mode, it will return two probabilities using the final softmax activation function contained in GeneCNN, the nongenic and genic class respectively (Shannon, 1948). The returned probabilities represent the likelihood that the entire 500 nucleotide length sample sequence is nongenic or genic. To generate predictions for a genome, the genome sequence must be split into 500 nucleotide length sequences. This requirement is due to the matrix-based calculations that are performed in a neural network, where since GeneCNN was trained on 500 nucleotide length sequences, predictions must also be made on 500 nucleotide length sequences (Figure 3-1).

To achieve a finer resolution for gene prediction rather than solely 500 nucleotide length sequences, a sliding window-based approach was developed. The sliding window technique is a scanning subset of a specified length where the subset is iteratively created then replaced by a new subset, allowing for the creation of smaller sequences to perform computational tasks within the total genome sequence. For the finest possible resolution, a sliding window stride of 1 is used, meaning that GeneCNN makes a prediction on a given 500 nucleotide sample sequence, then the sliding window is moved in its entirety 1 nucleotide downstream and GeneCNN makes a prediction on the newly created 500 nucleotide sample sequence. This sliding window approach is repeated for a desired length of the genome to yield gene predictions and the overall structure of genes for a given section of the genome (Figure 3-1).

Once sliding window predictions are made by GeneCNN, all predictions are used as support for nucleotide positions that were included in any given sliding window sample sequence. The average of these scores results in a final genic or nongenic

prediction per nucleotide of the genome, where contiguous sequences of genic predictions are collected. By applying a contiguity threshold, the number of genes must strictly be larger than the threshold, eliminating small gene predictions, that are likely false positive. Output of GeneCNN predictions is a FASTA file where predicted gene sequences are listed along with unique headers that state the nucleotide starting position of the predicted gene sequence (Figure 3-1).

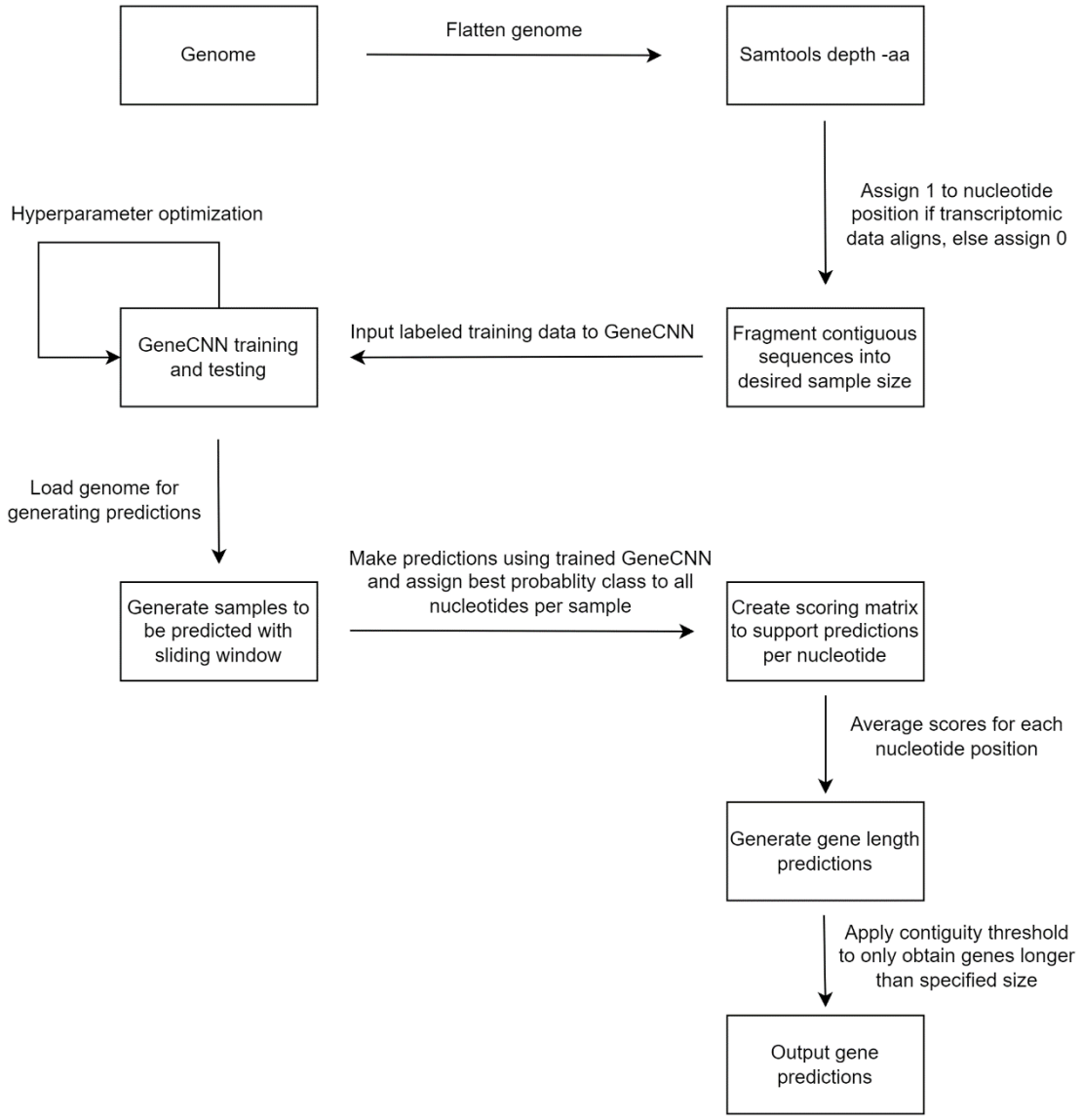

Figure 3-5. Visualization of the GeneCNN gene prediction workflow. A given genome is flattened by combining all separated sequences into a singular genome-length sequence. Samtools with option depth -aa is used to generate read depth per nucleotide where labels are assigned to the genome depending on whether transcriptomic data align to the nucleotide. All contiguous sequences of one label type are fragmented into the desired sample size and used as input to GeneCNN. GeneCNN is optimized through iterative testing including with an optimization algorithm. Once GeneCNN finishes training, the original genome file is loaded, and a sliding window approach is used to generate samples with the same size as samples used to train GeneCNN. All samples are predicted with GeneCNN and the best probability of either genic or nongenic is assigned to all nucleotides per sample. A scoring matrix is created which retains supporting predictions for each nucleotide in the genome per inclusion of each nucleotide in sliding window samples. The scores per nucleotide are averaged and gene length predictions are created

using contiguous predictions of genic nucleotides. A contiguity threshold is applied to remove low quality genic predictions and gene predictions are output.

3.2 AUGUSTUS

AUGUSTUS is a gene prediction tool that can be run in either *ab initio* mode or with the assistance of extrinsic data used as additional input (Hoff & Stanke, 2019). For a comparison with other gene prediction tools in this study, AUGUSTUS was run in *ab initio* mode. This comparison was made to elucidate the increase in performance due to generational improvements and species-specific model training compared to both GeneCNN and other gene prediction tools used in this study. When run in *ab initio* mode, AUGUSTUS utilizes species-specific parameters for its HMM implementation through pre-trained species-specific configuration files. If one of these configuration files are specified, then AUGUSTUS will accept input of only a genome and use the species-specific parameters to create gene predictions based on the statistical properties of the genome used for pre-training. These statistical properties of a particular genome are referred to as content sensors, which include hexamer frequency, sequence composition, and 3-base periodicity (Do & Choi, 2005; Fickett & Tung, 1992).

The modeling approach used by AUGUSTUS is a supervised approach, meaning that AUGUSTUS must have a carefully curated set of input data to train the model. The curated set of training data used for AUGUSTUS to make gene predictions should ideally be from the same organism that AUGUSTUS will generate gene predictions for. However, cross-species training is viable with AUGUSTUS so long as the two species are evolutionarily related. Hoff & Stanke (2019) verified the viability of cross-species

training by training AUGUSTUS with *Danio rerio* gene sequences then generating gene predictions for the *Gallus domesticus* genome and received only a 6.6% decrease in sensitivity and a 4.9% decrease in specificity, despite these two organisms diverging 429 million years ago (Kumar et al., 2017).

3.3 BRAKER3

A commonly used tool for gene prediction is the BRAKER software suite, of which the newest version is BRAKER3 (Gabriel et al., 2023). BRAKER3 functions by utilizing GeneMark-ETP and AUGUSTUS, both of which are HMM-based gene prediction algorithms (Bruna et al., 2023; Stanke et al., 2008). GeneMark-ETP, the newest iteration of GeneMark, has been updated since the release of BRAKER2 to integrate genomic, transcriptomic, and protein information over the course of model training and gene prediction.

The BRAKER3 pipeline can be provided with transcriptomic data, protein data, or both. If run with only one of these data types, BRAKER3 executes in ET mode, meaning that it uses the BRAKER1 algorithm. The BRAKER1 algorithm differs from the newest BRAKER3 algorithm in that it uses GeneMark-ET for half of the predictions made by BRAKER1, rather than the newest GeneMark-ETP. GeneMark-ET uses a statistical modeling approach for predicting genes that includes an extrinsic source of data which for this project is aligned unassembled transcriptomic data, where the extrinsic data improves GeneMark-ET model training by indicating intron positions and exon-noncoding region borders (Hoff et al., 2016). This use of extrinsic data allows GeneMark-ET to refine its parameters through self-training on the given genome. BRAKER3 opts to use the approach of unassembled transcriptomic data rather than

assembled transcriptomic data due to transcript assemblies being error prone, as transcript assemblers often fail to identify all exons (Steijger et al., 2013). Therefore, to avoid the propagation of these transcript assembly errors into gene predictions, unassembled transcriptomic data is used. Once GeneMark-ET has generated *ab initio* gene predictions, a subset of these genes are provided as input to AUGUSTUS, as AUGUSTUS requires an organism-specific training set of data to make predictions. AUGUSTUS then uses both the training set and the aligned unassembled transcriptomic data to make predictions. Once predictions are made by both GeneMark-ET and AUGUSTUS, BRAKER3 combines them into a singular prediction file which generates a more robust set of predictions than either of these gene prediction tools alone.

3.4 Fgenesh

Fgenesh is an older *ab initio* gene prediction tool, that was developed when dynamic programming was the prominent algorithm used for gene prediction (Salamov & Solovyev, 2000; Solovyev et al., 2006). Rather than following similar algorithms at the time, Fgenesh was instead built using HMMs to predict genes (Burge & Karlin, 1998). Fgenesh was not the first HMM-based gene prediction tool, however it innovated upon previously released tools by ascribing a stronger weight to signal sensors, such as gene start sites, than content sensors. This stronger weighting of signal sensors than content sensors was meant to reflect the biological significance of gene signals.

Fgenesh develops predictions in a similar way to AUGUSTUS, where implementation of its HMM-based model requires prior training on a specific organism to ascertain the values of content sensors for an organism. Similarly, Fgenesh also provides a number of species-specific training results to use for gene prediction for the provided

genomic sequence. Yao et al. (2005) evaluated the performance of Fgenesh trained on monocots to predict genes in *Zea mays,* where Fgenesh outperformed the four other gene prediction tools selected. This resulted in Fgenesh yielding an increase of 11% correct gene models than the second-best gene prediction tool despite Fgenesh not being trained with *Zea mays*-specific sequences. Output from Fgenesh differs from the other gene prediction tools by predicting a single gene region as the entire expanse between the transcription start site and the poly(A) tail, whereas the other three gene prediction tools focus on the coding regions and will at times predict two genes in the region where Fgenesh predicts one.

3.5 Comparing GeneCNN with other gene prediction tools

Due to the nature of GeneCNN predicting on entire 500 nucleotide length sequences, there is no simple comparison to make with HMM-based gene prediction tools other than through resulting gene predictions. The matrix calculations that occur in GeneCNN require specific data sizes, which are 500 nucleotide length sequences for the current iteration of GeneCNN, while HMM methods do not require this. However, the aforementioned sliding window approach yields the best comparison for gene predictions as it allows GeneCNN to iterate through the provided genomic sequence using 500 nucleotide length samples and a provided stride size. Utilizing the sliding window with specified stride size allows for the creation of a large scoring matrix where all probabilities for each 500 nucleotide sample are stored, in order of creation. A column average can be taken which yields a prediction result of genic or nongenic at the nucleotide level. Contiguous sequences of nucleotides with genic predictions are then defined as genes, where a length threshold is used to cull small, likely false positive,

genic sequences such as predictions of a singular nucleotide being genic. The length threshold for GeneCNN was defined to require 200 nucleotide length predicted sequences. The length of 200 nucleotide sequences was chosen as it should viably capture most gene sizes, including those with single exons (Atambayeva, 2007).

Once these full-length gene predictions are made using GeneCNN, the results can be compared to existing gene prediction tools, namely BRAKER3, AUGUSTUS, and Fgenesh. The comparison of GeneCNN with BRAKER3 is noteworthy due to the ongoing development and high accuracy of the BRAKER software suite in novel genomes with the combination approach of GeneMark-ET and AUGUSTUS (Hoff et al., 2019). Unique gene predictions made by GeneCNN compared to the other gene prediction tools can be validated with database results using BLAST, specifically blastx against the RefSeq non-redundant (nr) protein database, to confirm the presence of protein encoding genes at these positions (Altschul et al., 1990). Blastx converts predicted gene sequences into protein sequences and searches the nr protein database with the converted sequence. The presence of protein sequences for a given converted gene prediction supports that the prediction made by GeneCNN is likely true, particularly if the protein sequence found in the nr database is experimentally verified. The quality of BLAST results can be determined by bit score which is a value that represents the quality of query sequence to database sequence that is independent of the query sequence length and database size.

As a tool, BLAST can also be used for direct comparison rather than only for validating the performance comparisons between GeneCNN and the HMM-based gene prediction tools. By using blastx on the entire gene prediction set made by GeneCNN,

unique predictions made by GeneCNN that are not found in the nr protein database can be discovered. Additionally, BLAST results can provide support to all GeneCNN gene predictions that have significant similarity to protein sequences found in the nr protein database.

Methods

3.6 Creation of a dataset for gene prediction with GeneCNN

The same buffalograss genome that was used for training GeneCNN was used for generating gene predictions. Unlike when used to create the training dataset, the genome file used for predicting genes was unprocessed. The first 10 million nucleotides of the first pseudochromosome, or largest assembled scaffold, were used as the prediction dataset to allow for a smaller computational footprint in which to make comparisons among predicted genes made by each tool. This smaller computational footprint was necessary when using a stride size of 1, as the scoring matrix will cause the graphics processing unit to run out of available memory when used on the whole pseudochromosome at once. To prepare the 10 million nucleotide genome fragment for gene prediction using GeneCNN, the alphabetic representation of nucleotide bases was converted into a numeric representation. The 10 million nucleotide genome fragment was one-hot encoded as follows: ‘A’ to (1, 0, 0, 0), ‘C’ to (0, 1, 0, 0), ‘G’ to (0, 0, 1, 0), and ‘T’ to (0, 0, 0, 1).

3.7 Creation of a sliding window for GeneCNN predictions

A sliding window approach was used to iteratively predict genic sequences in the first 10 million nucleotides of the first pseudochromosome. The stride size used for this

sliding window was 1, so after creating a sample sequence of 500 nucleotide in length, another sample sequence was created by shifting one nucleotide downstream, creating a new 500 nucleotide length sample sequence. This sliding window process was done in genomic segments of length 16,000 to utilize the batch processing power of GeneCNN due to being built using TensorFlow. Predictions were generated for 500 samples per batch.

3.8 Defining GeneCNN gene predictions

Once predictions were generated for the entire 10 million nucleotide genome fragment, a scoring matrix was created. This scoring matrix was generated by assigning genic or nongenic predictions at each nucleotide position using the same sliding window approach with stride size of 1. Assignments of genic or nongenic were dependent on which received a higher probability when output from GeneCNN. The scoring matrix gives support to each nucleotide position for up to 500 different predictions made using this nucleotide in any given sliding window subset. The mean of each column was calculated, producing a singular value dependent on all predictions made using each nucleotide in the entire 10 million nucleotide genome fragment. To generate gene length predictions, a length threshold was created for all contiguous genic predicted nucleotides. The length threshold was defined to be 200 nucleotide length sequences, making the shortest gene predictions by GeneCNN 200 nucleotide in length. Output of GeneCNN is a FASTA file where each predicted gene is a nucleotide sequence that is associated with a header that defines the gene start position.

Gene predictions unique to GeneCNN in comparison to BRAKER3, AUGUSTUS, and Fgenesh were validated using NCBI BLAST, specifically blastx

against the nr protein sequence database (Altschul et al., 1990). The presence of protein sequences matching the converted gene predictions from GeneCNN yields support for protein encoding gene predictions. Further support is provided if the protein sequence results from BLAST have a high sequence similarity to the converted GeneCNN sequence or are experimentally validated.

Gene predictions unique to GeneCNN were also defined relative to BLAST results, where blastx was run on all genes predicted by GeneCNN. Running blastx on this gene prediction set from GeneCNN also yielded support by finding the number of genes that GeneCNN predicts that are also found in the nr protein sequence database.

3.9 Running BRAKER3

Unassembled transcriptomic data from NCBI BioProjects PRJNA237791, PRJNA297834, PRJNA578934, and PRJNA231727 was used for supporting BRAKER3 gene predictions. Each of these sets of transcriptomic data were globally aligned to the 10 million nucleotide genome fragment using BBMap (Marić, 2015). Aligned transcriptomic data were converted to binary format and sorted using Samtools (Li et al., 2009). BRAKER3 was run in BRAKER1 mode with input of the 10 million nucleotide genome fragment as the genome sequence to predict genes for, along with the aligned transcriptomic data in binary format as support. Output of BRAKER3 is a GTF file which lists the sequence feature, which includes gene, transcript, start codon, coding sequence, exon, intron, and stop codon. The BRAKER3 GTF output file also lists the start and end positions of each predicted sequence feature, and whether AUGUSTUS or GeneMark-ET predicted the feature.

3.10 Running AUGUSTUS

AUGUSTUS was run in *ab initio* mode, where only the query sequence, the 10 million nucleotide genome fragment, was used as input along with the pre-trained species identifier to use HMM parameters specific to a particular species' genome. The pre-trained species identifier used for running AUGUSTUS was *Zea mays*, the closest related species to buffalograss available in the provided configuration files from AUGUSTUS, as they both reside within the taxonomic family of Poaceae. Output from AUGUSTUS is a GTF file which lists the sequence feature, which only includes gene, transcript, and coding sequence, as well as the start and stop positions of the predicted sequence feature. AUGUSTUS also outputs the translated protein sequence for each predicted gene.

3.11 Running Fgenesh

Fgenesh was run using the 10 million nucleotide genome fragment as an input sequence, along with the pre-trained species name to use HMM parameters specific to the genome of a particular species. The pre-trained species used for running Fgenesh was *Setaria italica*, the closest related species to buffalograss available, as they both reside within the taxonomic family of Poaceae. Output from Fgenesh is in a custom format, where the sequence feature includes transcription start site, coding sequence, and poly-A tail. Additionally, Fgenesh output lists the start and end positions of all coding sequences, as well as the translated protein sequence for each predicted gene.

3.12 Creating total predicted gene counts

Total gene predictions for each gene prediction tool were used to generate counts on predictions made for the same gene for each combination of gene prediction tool. To

determine whether gene prediction tools predicted the same gene, the start and stop sites for predicted genes of one tool were used to compare against the start and stop sites for predicted genes of the three other tools in every combination. A leniency of 200 nucleotide length from both ends of a given predicted gene was used for determining if the predicted gene from one tool was the same as the predicted gene from another tool.

Results

3.13 GeneCNN gene predictions

GeneCNN predicted 3,015 genes within the 10 million nucleotide length genome fragment. To compare gene predictions made by GeneCNN to the HMM-based gene prediction tools, the number of uniquely predicted genes was calculated. GeneCNN uniquely predicted 1,089 gene predictions compared to BRAKER3, 1,535 compared to AUGUSTUS, and 478 compared to Fgenesh when using a leniency threshold of 200 nucleotides. When compared using BLAST, GeneCNN uniquely predicted 431 genes. To verify that the uniquely predicted genes from GeneCNN are truly gene sequences missed by the HMM-based gene prediction tools, blastx was used to query the nr protein database with uniquely predicted gene sequences from GeneCNN. Uniquely predicted genes from GeneCNN were found to be supported by the nr protein database at a range of 77.8% to 82.2% of the uniquely predicted genes from GeneCNN compared to each of the other gene prediction tools (Table 3-1). To further investigate the uniquely predicted genes from GeneCNN, one uniquely predicted gene compared to each tool was manually checked in the nr protein database results for experimental validation.

Table 3-1. Uniquely predicted genes from GeneCNN compared to BRAKER3, AUGUSTUS, and Fgenesh. All 3,015 predicted genes by GeneCNN were compared against gene predictions made by BRAKER3, AUGUSTUS, and Fgenesh to determine predictions unique to GeneCNN. Uniquely predicted genes from GeneCNN were queried against the NCBI nonredundant protein database using blastx. The number of uniquely predicted genes supported by protein sequences in the nr database are in the 'Unique in nr' column, where the percentage is calculated using values from the 'Uniquely predicted genes from GeneCNN' column. Uniquely predicted genes not found in the nr protein database are in the 'Unique not in nr' column, where the percentage is calculated using values from the 'Uniquely predicted genes from GeneCNN' column.

Tool comparison	Uniquely predicted genes from GeneCNN	Unique in nr	Unique not in nr
GeneCNN vs BRAKER3	1,089	895 (82.2%)	194 (17.8%)
GeneCNN vs AUGUSTUS	1,535	1,233 (80.3%)	302 (19.7%)
GeneCNN vs Fgenesh	478	372 (77.8%)	106 (22.2%)

A unique prediction made by GeneCNN in comparison to BRAKER3 is found starting at nucleotide position 4561 and ending at 5009 on the first pseudochromosome of the buffalograss genome. When searched against the nr database using blastx, a number of similar proteins are returned. The top three of which are derived from the organisms *Panicum hallii*¸ *Panicum virgatum,* and *Zea mays,* all of which belong to the same family as buffalograss of Poaceae. All three of these protein sequences return a bit score ranging from 164 to 167 with sequence identity ranging from 81.91% to 84.78%. However, all three of these protein sequences found in the nr database are predicted proteins which were generated by the NCBI tool Gnomon. There is also an experimentally verified protein in the results from searching the nr protein database, namely a hAT transposon superfamily protein with the accession CAA0807048.1 from the organism *Striga*

hermonthica, that returned a bit score of 132 and sequence identity of 75% (Qiu et al., 2022).

One unique prediction made by GeneCNN in comparison to AUGUSTUS is found starting at nucleotide position 180482 and ending at 181248 on the first pseudochromosome of the buffalograss genome. When searched against the nr database using blastx, the top three results are from the organisms *Panicum miliaceum*, *Oryza sativa*, and *Digitaria exilis*, all of which are in the same family of Poaceae as buffalograss. The bit scores and sequence identities for these three protein results range from 219 to 228, and 88.5% to 93.64% respectively. All three of these resulting proteins are gene predictions from other gene predictions tools that were submitted to the nr database. However, there are a number of experimentally verified proteins in the nr database results given the unique prediction by GeneCNN as a query. The most notable of which is the DExH-box ATP-dependent RNA helicase DExH12 isolated from *Zea mays* which has the accession ONM60047.1 (Jiao et al., 2017). This nr database result has a bit score of 225 and a sequence identity of 91.15%.

A unique prediction made by GeneCNN in comparison to Fgenesh is found starting at nucleotide position 237433 and ending at position 242295 on the first pseudochromosome of the buffalograss genome. When searched against the nr database using blastx, the top three results are hypothetical proteins predicted in the organisms *Eragrostis curvula*, *Panicum virgatum*, and *Panicum hallii*. These nr database results yielded bit scores ranging from 1311 to 1343, and sequence identities ranging from 88.66% to 92.49%. The fourth best nr database result is the experimentally verified leucine-rich repeat receptor-like protein kinase PXL2 isolated from *Panicum miliaceum*

with the accession of RLM80341.1 (Zou et al., 2019). This experimentally verified nr database result has a bit score of 1310 and a sequence identity of 89.42% when found with the unique prediction generated by GeneCNN compared to Fgenesh.

3.14 BRAKER3, AUGUSTUS, and Fgenesh gene predictions

BRAKER3 predicted 2,068 genes within the 10 million nucleotide genome fragment when using aligned unassembled transcriptomic data as support. AUGUSTUS predicted 1,865 genes within the 10 million nucleotide genome fragment when using the *Zea mays*-specific genome parameters in *ab initio* mode. Fgenesh predicted 2,308 genes within the 10 million nucleotide genome fragment when using the *Setaria italica*-specific genome parameters.

3.15 Comparison of gene predictions with all tools

To compare how well BRAKER3, AUGUSTUS, and Fgenesh performed in comparison to GeneCNN, the number of unique predictions made by these gene prediction tools was calculated. BRAKER3 uniquely predicted 1,014 genes compared to GeneCNN, AUGUSTUS uniquely predicted 1,305 genes compared to GeneCNN, and Fgenesh did not predict any unique genes when compared to GeneCNN (Table 3-2). While Fgenesh did not predict any unique genes compared to GeneCNN, the number of predicted genes shared by all gene prediction tool combinations including Fgenesh is high due to how Fgenesh predicts genes. A gene predicted with Fgenesh is denoted as the entire region spanning from a transcription start site to the poly(A) tail, whereas the other gene prediction methods focus on coding regions, which can lead to two predictions made by the other tools spanning one prediction made by Fgenesh.

Table 2-2. Uniquely predicted genes from BRAKER3, AUGUSTUS, and Fgenesh, compared to GeneCNN. Uniquely predicted genes from each tool were queried against the NCBI nonredundant protein database using blastx. The number of uniquely predicted genes supported by protein sequences in the nr database are in the 'Unique in nr' column, where the percentage is calculated using values from the 'Uniquely predicted genes compared to GeneCNN' column. Uniquely predicted genes not found in the nr protein database are in the 'Unique not in nr' column, where the percentage is calculated using values from the 'Uniquely predicted genes compared to GeneCNN' column.

Tool comparison	Total predicted genes compared to GeneCNN	Uniquely predicted genes compared to GeneCNN	Unique in nr	Unique not in nr
BRAKER3 vs GeneCNN	2,068	1,014	901 (88.9%)	113 (11.1%)
AUGUSTUS vs GeneCNN	1,865	1,305	1,234 (94.6%)	71 (5.4%)
Fgenesh vs GeneCNN	2,308	0	N/A	N/A

Gene predictions made by GeneCNN that are shared with gene predictions made by the other three HMM-based tools are relatively large at a minimum value of 40% of all genes predicted by GeneCNN, which supports that GeneCNN is accurately predicting genes as the majority of predictions made by any given gene prediction tool should also be predicted by other gene prediction tools (Table 3-3). However, it can also be seen that AUGUSTUS gene predictions are somewhat divergent compared to GeneCNN which affects all gene prediction tool combinations that include AUGUSTUS. As such, this divergence in gene prediction of AUGUSTUS may be due to the maize-specific genome parameters used for its HMM model.

Table 3-3. The number of gene predictions made by GeneCNN that are supported by BRAKER3, AUGUSTUS, Fgenesh, or some combination of the three. The initial 'B' represents BRAKER3, 'A' represents AUGUSTUS, and 'F' represents Fgenesh. Numeric values are the number of gene predictions out of 3,015 total genes predicted by GeneCNN that are supported by at least one other tool. A leniency of 200 nucleotides in length from both ends of predicted genes was included in determining whether the predictions made by GeneCNN were the same as genes predicted by the other three tools.

Gene prediction tool combinations	Number of gene predictions made by GeneCNN supported by other tools
B, A, F	1220 (40.5%)
B, A	1249 (41.4%)
B, F	1786 (59.2%)
A, F	1434 (47.6%)
B	1940 (64.3%)
A	1480 (49.1%)
F	2537 (84.1%)

BRAKER3, AUGUSTUS, and Fgenesh were also compared to each other to see how successful they were at predicting the same genes (Table 3-4; Table 3-5; Table 3-6). Comparisons of AUGUSTUS and Fgenesh to BRAKER3 resulted in a low percentage of 27.8% shared gene predictions, increasing to 59% shared gene predictions when comparing Fgenesh to BRAKER3 (Table 3-4). Comparisons of BRAKER3 and Fgenesh to AUGUSTUS resulted in a large percentage of shared gene predictions ranging from 75% to 87.5% (Table 3-5). Comparisons of BRAKER3 and AUGUSTUS to Fgenesh resulted in a low percentage of shared gene predictions, ranging from 10.4% to 32.5%, where best support comes from comparing only BRAKER3 to Fgenesh (Table 3-6).

Table 3-4. The number of gene predictions made by BRAKER3 that are supported by AUGUSTUS, Fgenesh, or the combination of AUGUSTUS and Fgenesh. The initial 'A' represents AUGUSTUS and 'F' represents Fgenesh. Numeric values are the number of gene predictions out of 2,068 total genes predicted by BRAKER3 that are supported by at least one other tool. A leniency of 200 nucleotides in length from both ends of predicted genes was included in determining whether the predictions made by BRAKER3 were the same as genes predicted by the other three tools.

Gene prediction tool combinations	Number of gene predictions made by BRAKER3 that are supported by AUGUSTUS and Fgenesh
A, F	574 (27.8%)
A	623 (30.1%)
F	1220 (59%)

Table 3-5. The number of gene predictions made by AUGUSTUS that are supported by BRAKER3, Fgenesh, or the combination of BRAKER3 and Fgenesh. The initial 'B' represents BRAKER3 and 'F' represents Fgenesh. Numeric values are the number of gene predictions out of 1,865 total genes predicted by AUGUSTUS that are supported by at least one other tool. A leniency of 200 nucleotides in length from both ends of predicted genes was included in determining whether the predictions made by AUGUSTUS were the same as genes predicted by the other three tools.

Gene prediction tool combinations	Number of gene predictions made by AUGUSTUS that are supported by BRAKER3 and Fgenesh
B, F	1400 (75.1%)
B	1503 (80.6%)
F	1631 (87.5%)

Table 3-6. The number of gene predictions made by Fgenesh that are supported by BRAKER3, AUGUSTUS, or the combination of BRAKER3 and AUGUSTUS. The initial 'B' represents BRAKER3 and 'A' represents AUGUSTUS. Numeric values are the number of gene predictions out of 2,308 total genes predicted by Fgenesh that are supported by at least one other tool. A leniency of 200 nucleotides in length from both ends of predicted genes was included in determining whether the predictions made by Fgenesh were the same as genes predicted by the other three tools.

Gene prediction tool combinations	Number of gene predictions made by Fgenesh that are supported by BRAKER3 and AUGUSTUS
B, A	241 (10.4%)
B	750 (32.5%)
A	320 (13.9%)

Comparisons of AUGUSTUS, BRAKER3, and Fgenesh to each other confirm that the gene predictions generated by AUGUSTUS are divergent, as comparisons including AUGUSTUS to both BRAKER3 and Fgenesh result in very low percentages of shared predictions (Table 3-4; Table 3-6). Additionally, the comparisons of BRAKER3 and Fgenesh to AUGUSTUS yielded very high percentages of shared predictions (Table 3-5). This inverse relationship in comparisons to and with AUGUSTUS states that the gene predictions made by AUGUSTUS are very small and likely fragmented genes, such that 75-87.5% of AUGUSTUS predictions can be found within the length of genes predicted by BRAKER3 and Fgenesh, while only upwards of 30% of BRAKER3 and Fgenesh predictions can be found within the length of genes predicted by AUGUSTUS. This low gene prediction performance of AUGUSTUS means that each prediction made by AUGUSTUS potentially only represents a single exon, while GeneCNN, BRAKER3, and Fgenesh, are generating gene length predictions. The poor comparisons including AUGUSTUS also suggest that the number of GeneCNN predictions supported by

BRAKER3 and Fgenesh, at a range of 59.2% to 84.1% shared gene predictions, are much more indicative of the overall performance of GeneCNN, as the majority of predictions made by GeneCNN are also found by two other gene prediction tools (Table 3-3).

When compared directly using BLAST, GeneCNN predicted genes were supported by 85.7% of the BLAST results. Running blastx on the entire gene prediction set by GeneCNN yielded significant similarity results in the nr protein sequence database for 2,584 of the total 3,015 genes predicted by GeneCNN.

Discussion

The predictive power of GeneCNN when trained on a novel genome was demonstrated through the comparison of gene predictions made with BRAKER3, AUGUSTUS, and Fgenesh. The number of unique genes predicted by GeneCNN were comparable to BRAKER3 and AUGUSTUS when both were compared against GeneCNN. GeneCNN uniquely predicted 1,089 genes compared to BRAKER3, 1,535 compared to AUGUSTUS, and 478 compared to Fgenesh. Meanwhile, BRAKER3 and AUGUSTUS uniquely predicted 1,014 and 1,305 genes compared to GeneCNN respectively. A singular unique gene prediction for each comparison of GeneCNN to another tool were demonstrated to have at least one experimentally validated gene that was missing from the predictions made by BRAKER3, AUGUSTUS, and Fgenesh. As BRAKER3 is one of the highest performing gene prediction tools currently available, especially for use in novel genomes, the resulting performance by GeneCNN in comparison to BRAKER3 is significant. In the case of the experimentally validated gene predicted by GeneCNN and not by Fgenesh, the bit score result was 1310, one of the highest scores observed in unique predictions checked with blastx.

The number of predictions alone, however, does not fully represent the quality of these predictions. One known area where GeneCNN is known to lose prediction power is in genes with long introns, as it has previously predicted two genes within the span of one predicted gene by BRAKER3. This is likely due to a number of factors which include the kernel size used during GeneCNN training, the 500 nucleotide length sample size, and the sliding window stride size used during gene prediction. The 500 nucleotide length sample size alone should not constitute much of the loss in prediction power however, as the average intron size in *Setaria italica*, a species in the same taxonomic family as buffalograss, is 442 nucleotides in length (Zhang et al., 2012). Therefore, the 500 nucleotide length sample size used by GeneCNN will capture most genes in a single prediction. GeneCNN predictions could potentially be improved by changing the kernel size during training, even if it generates a lower training performance, as this directly influences the spatial dependence of a given datapoint on its surrounding datapoints. GeneCNN predictions could also potentially be improved by modifying the sliding window stride size, as this generates the amount of support for assigning any given nucleotide position as genic. By increasing the sliding window stride size, there will be an overall loss in support per nucleotide position but there will be a longer range in support for whole genic sequences. Likely some combination of changing the kernel size during GeneCNN training and changing the sliding window stride size for creating whole gene sequences will yield the highest performance. Both the kernel size and stride size should be considered before modifying the sample size length for GeneCNN.

The resulting genes predicted by GeneCNN could likely be refined by incorporating transcriptomic data into more than solely creating class labels for training

and test datasets. One of the advantages that the HMM-based gene prediction tools have over GeneCNN is that they have a stronger incorporation of signal sensors. These signal sensors include genomic facets such as transcription start sites as well as start and stop codons. In comparison, GeneCNN defines genic start and end sites through the iterative incorporation of spatially dependent information into singular datapoints. So, while some measure of signal sensors are incorporated into GeneCNN natively through convolutions and dimensionality reductions, a stricter inclusion of these signal sensors may better refine the ends of genic predictions made by GeneCNN.

A major benefit of GeneCNN is the relatively low amount of data required to obtain performance rivaling BRAKER3, AUGUSTUS, and Fgenesh. The buffalograss genome was used as a foundation for the training and test datasets, with publicly available transcriptomic data from four different BioProjects used to generate genic labels for the training and test data. Once trained on the buffalograss genome with supporting information, the buffalograss genome was then reused for gene prediction. As GeneCNN is composed of a neural network for its modelling algorithm, a larger amount of supporting data or application of GeneCNN to a larger, more complex genome than the diploid buffalograss genome should increase the performance of GeneCNN even further compared to other gene prediction tools. This is due to the potential performance increases in neural networks as datasets grow larger and more complex, where their nonlinear nature allows for better generalizations of complex datasets than other statistical models (Korotcov et al., 2017).

Given that this is the first major iteration of GeneCNN and therefore has potential to be further refined, its performance compared to other gene prediction tools is

noteworthy. There are a higher number of genes uniquely predicted by GeneCNN compared to the other tools, where an average of 80% of unique predictions by GeneCNN were supported by protein data in the NCBI nr database (Table 3-1). Additionally, the majority of the total GeneCNN predicted genes were supported by other tools, where 85.7% of GeneCNN predicted genes were supported by BLAST results, while BRAKER3 and Fgenesh gene predictions supported 64.3% to 84.1% of the total GeneCNN predicted gene set (Table 3-3). Though there is some bias in the comparisons of uniquely predicted genes, multiple uniquely predicted genes generated from GeneCNN compared to BRAKER3, AUGUSTUS, and Fgenesh were queried against the nr protein database using blastx and were shown to be experimentally validated protein encoding genes. As such, there is value in the creation of the CNN-based gene prediction tool GeneCNN, as the spatial dependence informed gene predictions capture genes that are not found in the gene prediction sets generated by the HMM-based tools BRAKER3, AUGUSTUS, and Fgenesh.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2016). TensorFlow: Large-Scale machine learning on heterogeneous distributed systems. *ArXiv Preprint ArXiv:1603.04467*. http://arxiv.org/abs/1603.04467

Altschul, S., Gish, W., Miller, W., Myers, E., & Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, *215*(3), 403–410.

Atambayeva, S. A., Khailenko, V. A., & Ivashchenko, A. T. (2008). Intron and exon length variation in *Arabidopsis*, rice, nematode, and human. *Molecular Biology*, *42*(2), 312–320. https://doi.org/10.1134/S0026893308020180

Bruna, T., Lomsadze, A., & Borodovsky, M. (2023). GeneMark-ETP: Automatic gene finding in eukaryotic genomes in consistency with extrinsic data. *BioRxiv : The Preprint Server for Biology*. https://doi.org/10.1101/2023.01.13.524024

Burge, C. B., & Karlin, S. (1998). Finding the genes in genomic DNA. *Current Opinion in Structural Biology*, *8*, 346–354. http://biomednet.com/elecref/O959440XO0800346

Do, J. H., & Choi, D. K. (2006). Computational Approaches to Gene Prediction. *Korean Journal of Microbiology*, *44*(2), 137–144.

Fickett, J. W., & Tung, C. S. (1992). Assessment of protein coding measures. *Nucleic Acids Research*, *20*(24), 6441–6450. https://doi.org/10.1093/nar/20.24.6441

Gabriel, L., Brůna, T., Hoff, K. J., Ebel, M., Lomsadze, A., Borodovsky, M., & Stanke, M. (2023). BRAKER3: Fully automated genome annotation using RNA-Seq and protein evidence with GeneMark-ETP, AUGUSTUS and TSEBRA. *BioRxiv : The Preprint Server for Biology*. https://doi.org/10.1101/2023.06.10.544449

Henderson, J., Salzberg, S., & Fasman, K. H. (1997). Finding genes in DNA with a Hidden Markov Model. *Journal of Computational Biology*, *4*(2), 127–141. https://doi.org/10.1089/cmb.1997.4.127

Hoff, K. J., Lange, S., Lomsadze, A., Borodovsky, M., & Stanke, M. (2016). BRAKER1: Unsupervised RNA-Seq-based genome annotation with GeneMark-ET and AUGUSTUS. *Bioinformatics*, *32*(5), 767–769. https://doi.org/10.1093/bioinformatics/btv661

Hoff, K. J., & Stanke, M. (2019). Predicting genes in single genomes with AUGUSTUS. *Current Protocols in Bioinformatics*, *65*(1). https://doi.org/10.1002/cpbi.57

Jiao, Y., Peluso, P., Shi, J., Liang, T., Stitzer, M. C., Wang, B., Campbell, M. S., Stein, J. C., Wei, X., Chin, C. S., Guill, K., Regulski, M., Kumari, S., Olson, A., Gent, J., Schneider, K. L., Wolfgruber, T. K., May, M. R., Springer, N. M., … Ware, D. (2017). Improved maize reference genome with single-molecule technologies. *Nature*, *546*(7659), 524–527. https://doi.org/10.1038/nature22971

Korotcov, A., Tkachenko, V., Russo, D. P., & Ekins, S. (2017). Comparison of deep learning with multiple machine learning methods and metrics using diverse drug discovery data sets. *Molecular Pharmaceutics*, *14*(12), 4462–4475. https://doi.org/10.1021/acs.molpharmaceut.7b00578

Kumar, S., Stecher, G., Suleski, M., & Blair Hedges, S. (2017). TimeTree: A resource for timelines, timetrees, and divergence times. *Molecular Biology and Evolution*, *34*(7), 1812–1819. https://doi.org/10.1093/MOLBEV/MSX116

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics*, *25*(16). https://doi.org/10.1093/bioinformatics/btp352

Marić, J. (2015). Long read RNA-Seq mapper. *University of Zagreb-Faculty of Electrical Engineering and Computing-Master Thesis No*, *1005*.

Qiu, S., Bradley, J. M., Zhang, P., Chaudhuri, R., Blaxter, M., Butlin, R. K., & Scholes, J. D. (2022). Genome-enabled discovery of candidate virulence loci in *Striga hermonthica*, a devastating parasite of African cereal crops. *New Phytologist*, *236*(2), 622–638. https://doi.org/10.1111/nph.18305

Renwick, J. H. (1971). The mapping of human chromosomes. *Annual Review of Genetics*, *5*(1), 81–120. https://doi.org/10.1146/annurev.ge.05.120171.000501

Salamov, A. A., & Solovyev, V. v. (2000). Ab initio gene finding in *Drosophila* genomic DNA methods. *Genome Research*, *10*(4), 516–522. https://doi.org/10.1101/gr.10.4.516

Shannon, C. E. (1948). A Mathematical Theory of Communication. *The Bell System Technical Journal*, *27*(3), 379–423.

Solovyev, V., Kosarev, P., Seledsov, I., & Vorobyev, D. (2006). Automatic annotation of eukaryotic genes, pseudogenes and promoters. *Genome Biology*, *7*(1), 1–12. https://doi.org/10.1186/gb-2006-7-s1-s10

Stanke, M., Diekhans, M., Baertsch, R., & Haussler, D. (2008). Using native and syntenically mapped cDNA alignments to improve de novo gene finding. *Bioinformatics*, *24*(5), 637–644. https://doi.org/10.1093/bioinformatics/btn013

Steijger, T., Abril, J. F., Engström, P. G., Kokocinski, F., Akerman, M., Alioto, T., Ambrosini, G., Antonarakis, S. E., Behr, J., Bertone, P., Bohnert, R., Bucher, P., Cloonan, N., Derrien, T., Djebali, S., Du, J., Dudoit, S., Gerstein, M., Gingeras, T. R., … Zhang, M. Q. (2013). Assessment of transcript reconstruction methods for RNA-Seq. *Nature Methods*, *10*(12), 1177–1184. https://doi.org/10.1038/nmeth.2714

Yao, H., Guo, L., Fu, Y., Borsuk, L. A., Wen, T. J., Skibbe, D. S., Cui, X., Scheffler, B. E., Cao, J., Emrich, S. J., Ashlock, D. A., & Schnable, P. S. (2005). Evaluation of five ab initio gene prediction programs for the discovery of maize genes. *Plant Molecular Biology*, *57*(3), 445–460. https://doi.org/10.1007/s11103-005-0271-1

Zakharov, I. A. (2010). Location of genes in chromosomes: Random or not? *Russian Journal of Genetics*, *46*(10), 1165–1172. https://doi.org/10.1134/S1022795410100066

Zhang, G., Liu, X., Quan, Z., Cheng, S., Xu, X., Pan, S., Xie, M., Zeng, P., Yue, Z., Wang, W., Tao, Y., Bian, C., Han, C., Xia, Q., Peng, X., Cao, R., Yang, X., Zhan, D., Hu, J., … Wang, J. (2012). Genome sequence of foxtail millet (*Setaria italica*) provides insights into grass evolution and biofuel potential. *Nature Biotechnology*, *30*(6), 549–554. https://doi.org/10.1038/nbt.2195

Zou, C., Li, L., Miki, D., Li, D., Tang, Q., Xiao, L., Rajput, S., Deng, P., Peng, L., Jia, W., Huang, R., Zhang, M., Sun, Y., Hu, J., Fu, X., Schnable, P. S., Chang, Y., Li, F., Zhang, H., … Zhang, H. (2019). The genome of broomcorn millet. *Nature Communications*, *10*(1), 436. https://doi.org/10.1038/s41467-019-08409-5

APPENDIX A: SUPPLEMENTAL FIGURES

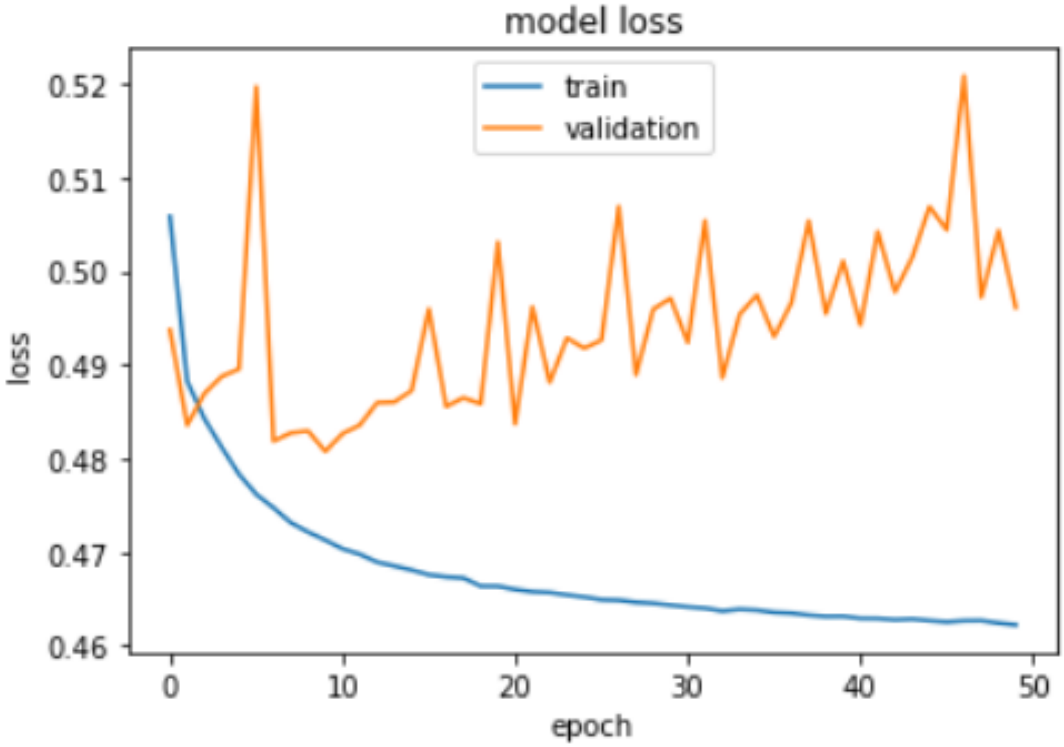

Figure A-1. Historical graph result of the model loss over time using the '200 nt' iteration of GeneCNN. Loss is shown over a training period of 50 epochs with both training and validation datasets. Model overfitting can be seen through the increasing degree of loss in the validation data during the training period.

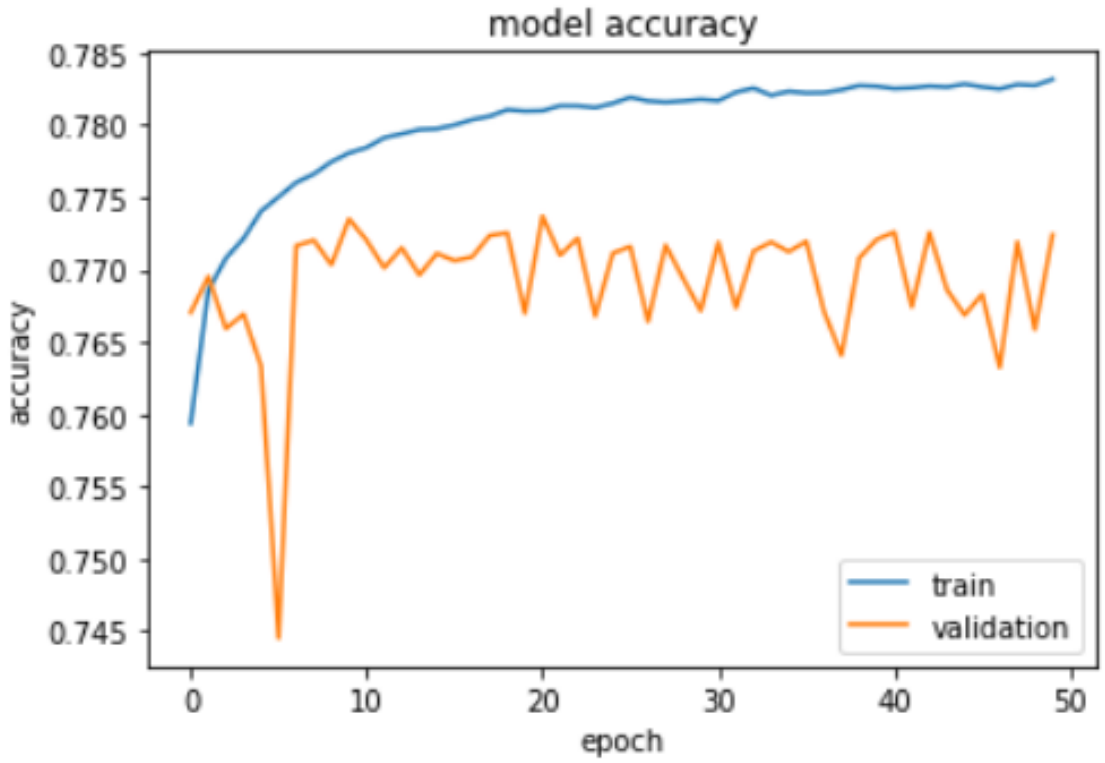

Figure A-2. Historical graph result of the model accuracy over time using the '200 nt' iteration of GeneCNN. Accuracy is shown over a period of 50 epochs with both training and validation datasets. A small degree of overfitting may be present but there is variance in the validation data once it plateaus at roughly 77%.

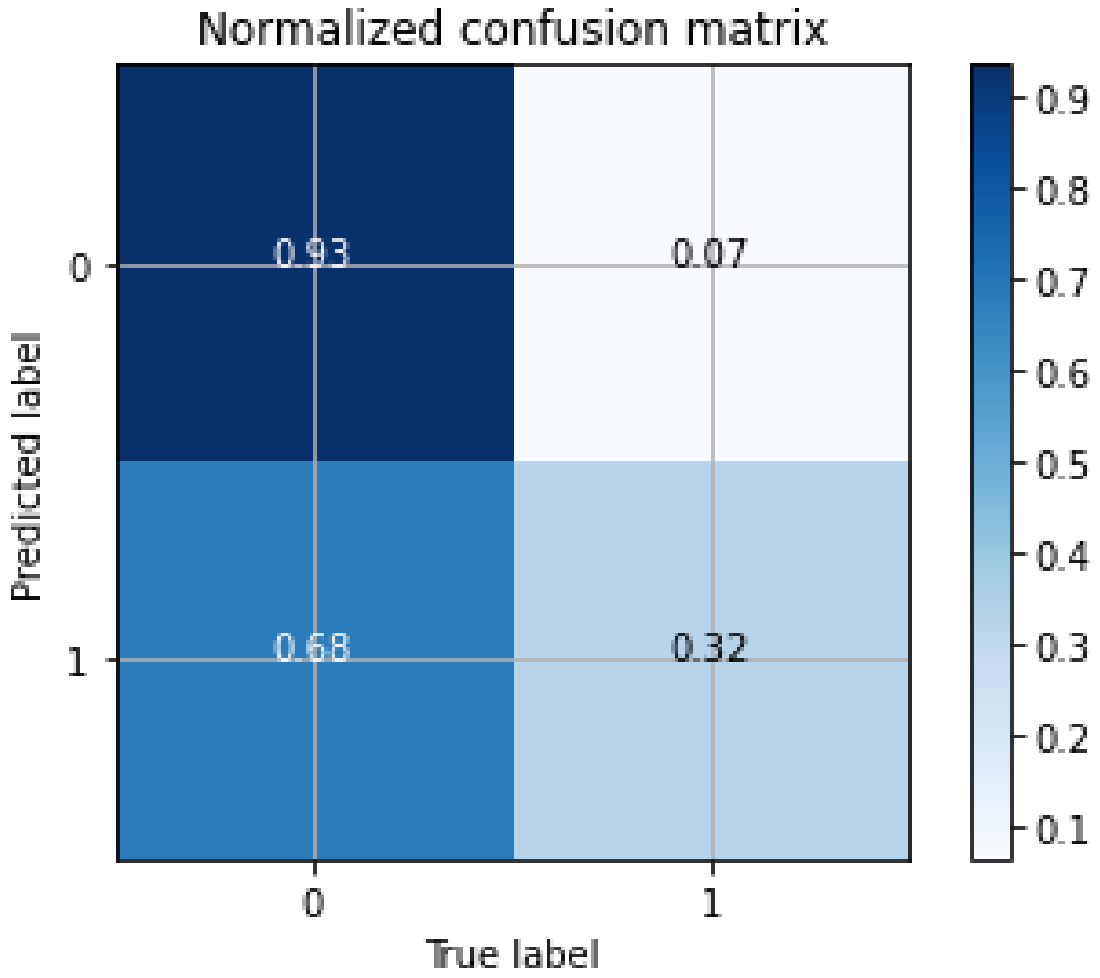

Figure A-3. Historical confusion matrix for the '200 nt' iteration of GeneCNN. The nongenic class prediction accuracy was 93%, found in the top left quadrant, while the genic class prediction accuracy was 32%, found in the bottom right quadrant. The false positive rate was 68%, found in the bottom left quadrant, while the false negative rate was 7%, found in the top right quadrant.

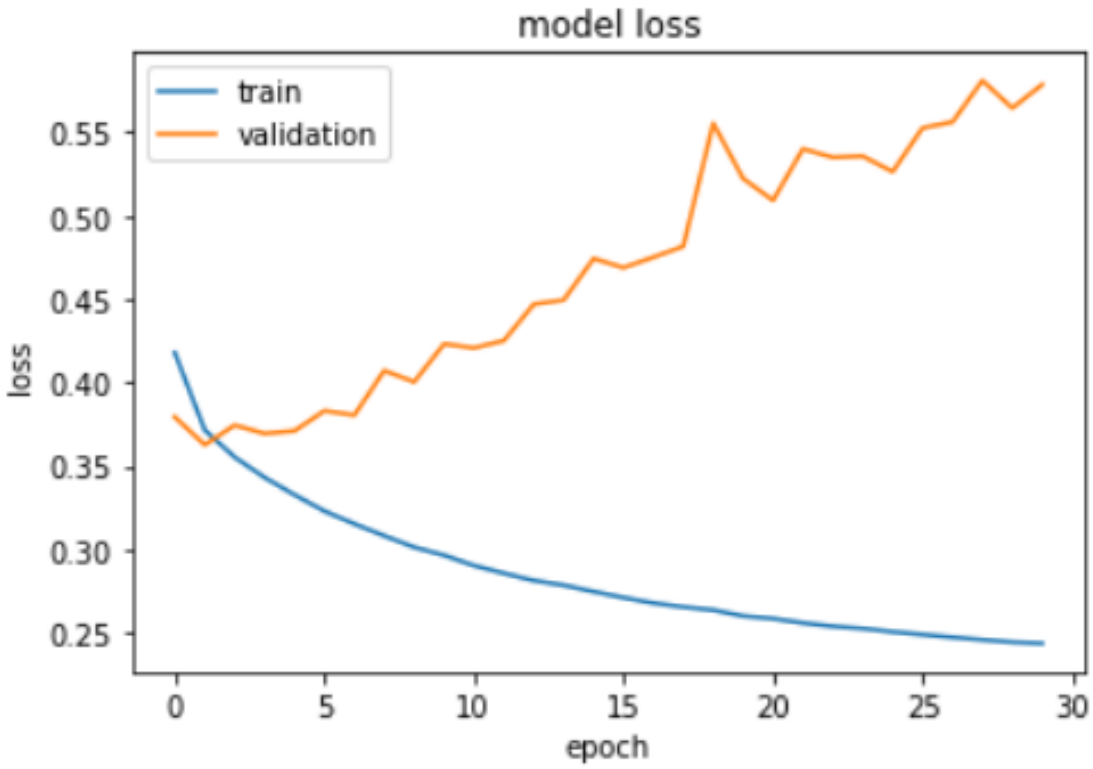

Figure A-4. Historical graph result of the model loss over time using the '1 Conv Layer' iteration of GeneCNN. Loss is shown over a training period of 30 epochs with both training and validation datasets. Model overfitting can be seen through the increasing degree of loss in the validation data during the training period.

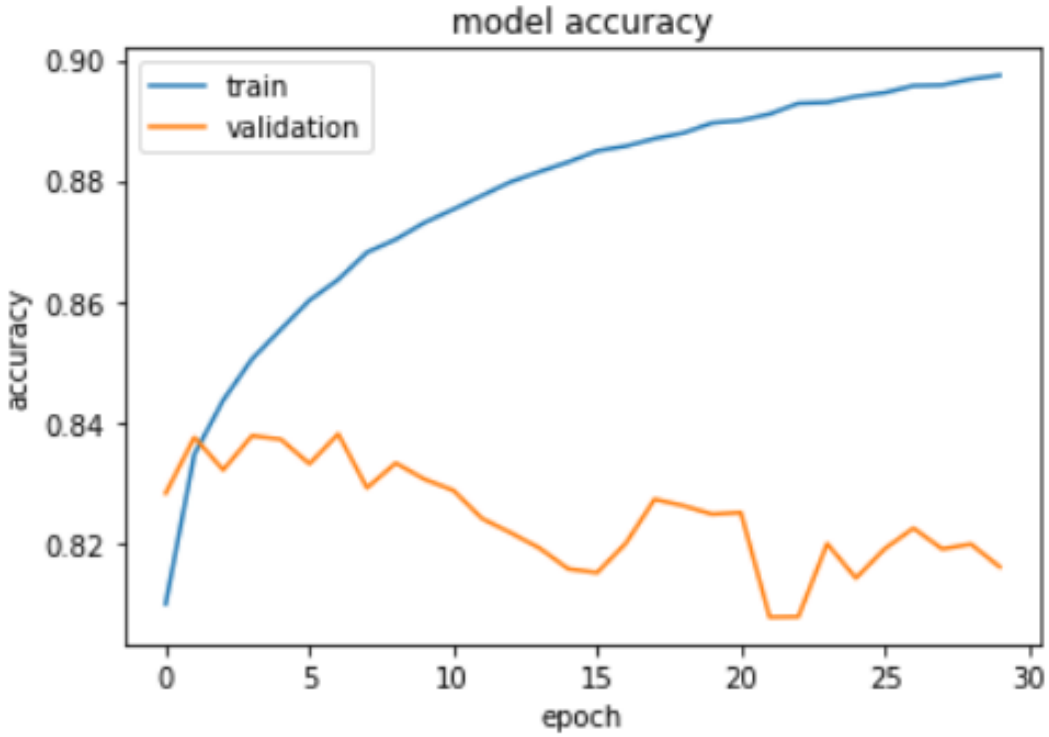

Figure A-5. Historical graph result of the model accuracy over time using the '1 Conv Layer' iteration of GeneCNN. Accuracy is shown over a period of 30 epochs with both training and validation datasets. Model overfitting can be seen through the decreasing accuracy in the validation data relative to the training data during the training period.

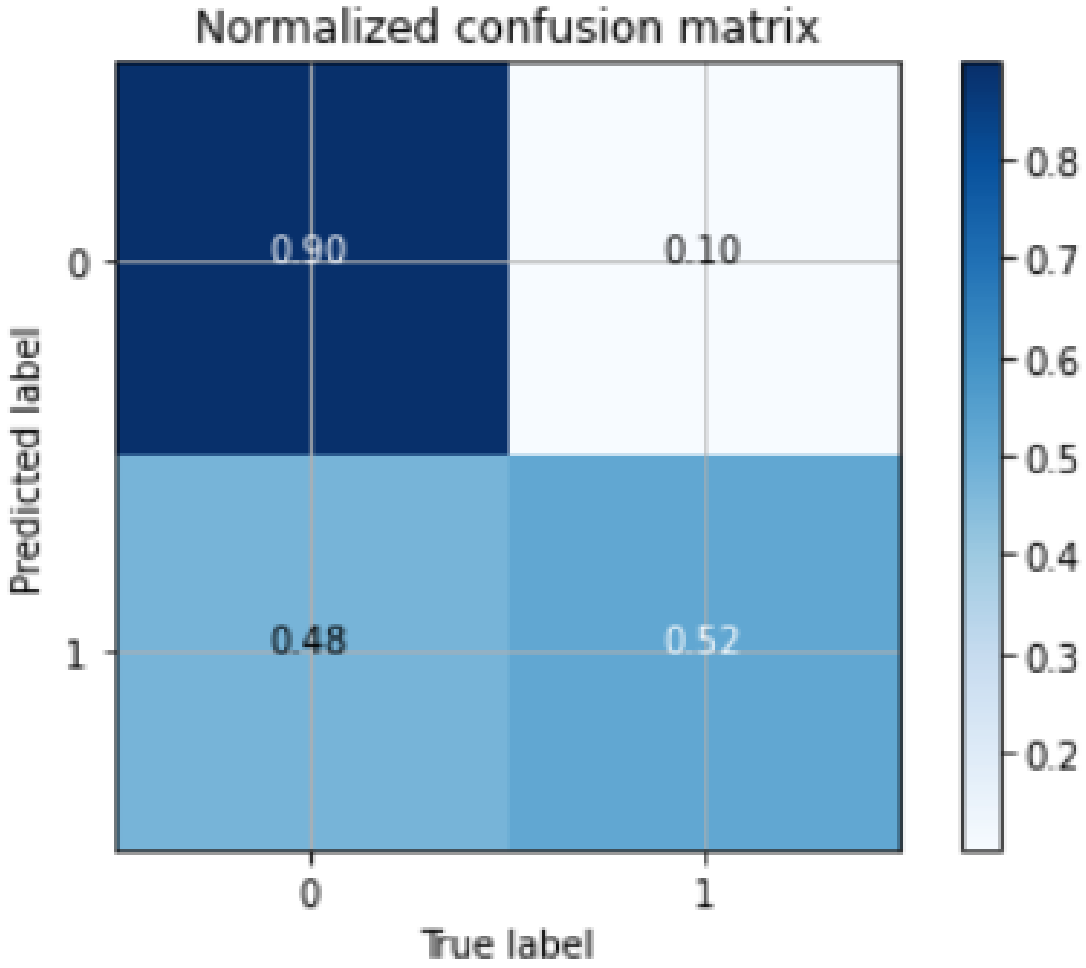

Figure A-6. Historical confusion matrix for the '1 Conv Layer' iteration of GeneCNN. The nongenic class prediction accuracy was 90%, found in the top left quadrant, while the genic class prediction accuracy was 52%, found in the bottom right quadrant. The false positive rate was 48%, found in the bottom left quadrant, while the false negative rate was 10%, found in the top right quadrant.

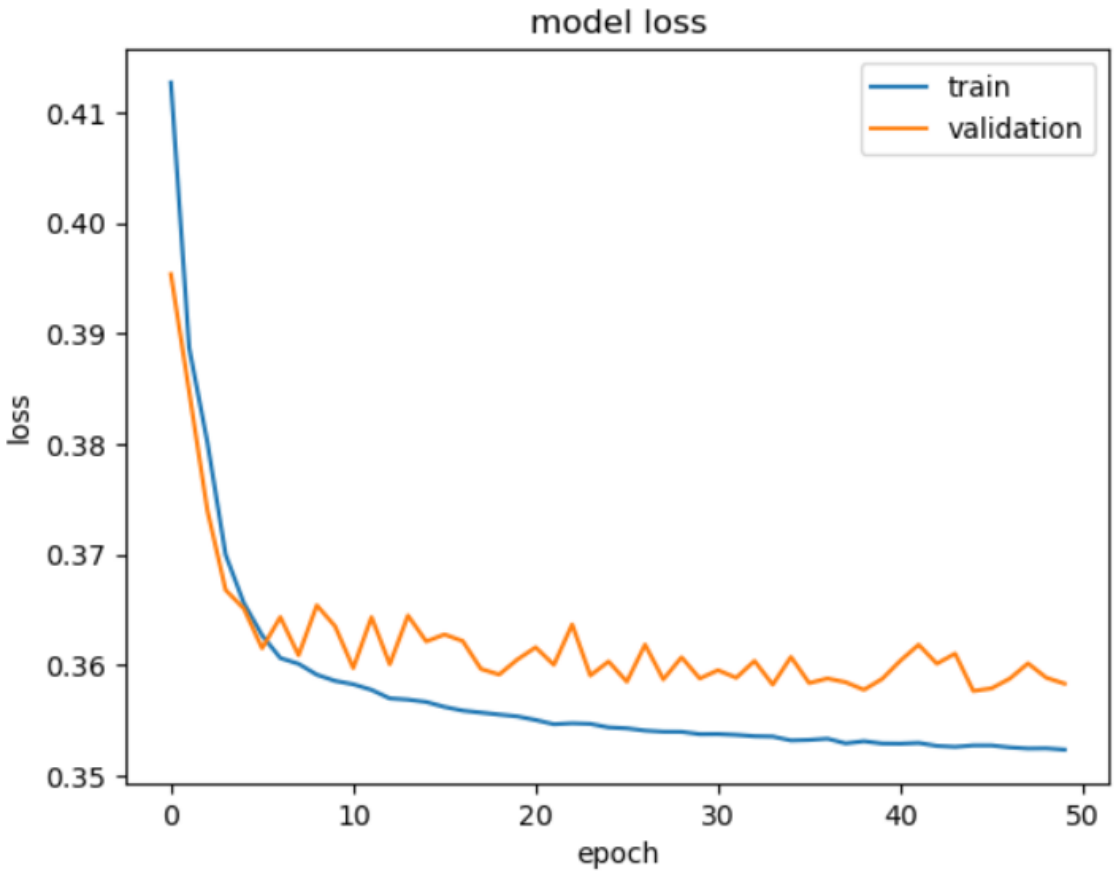

Figure A-7. Historical graph result of the model loss over time using the '4.9:1 Class Ratio' iteration of GeneCNN. Loss is shown over a training period of 50 epochs with both training and validation datasets. Model overfitting is not evident here.

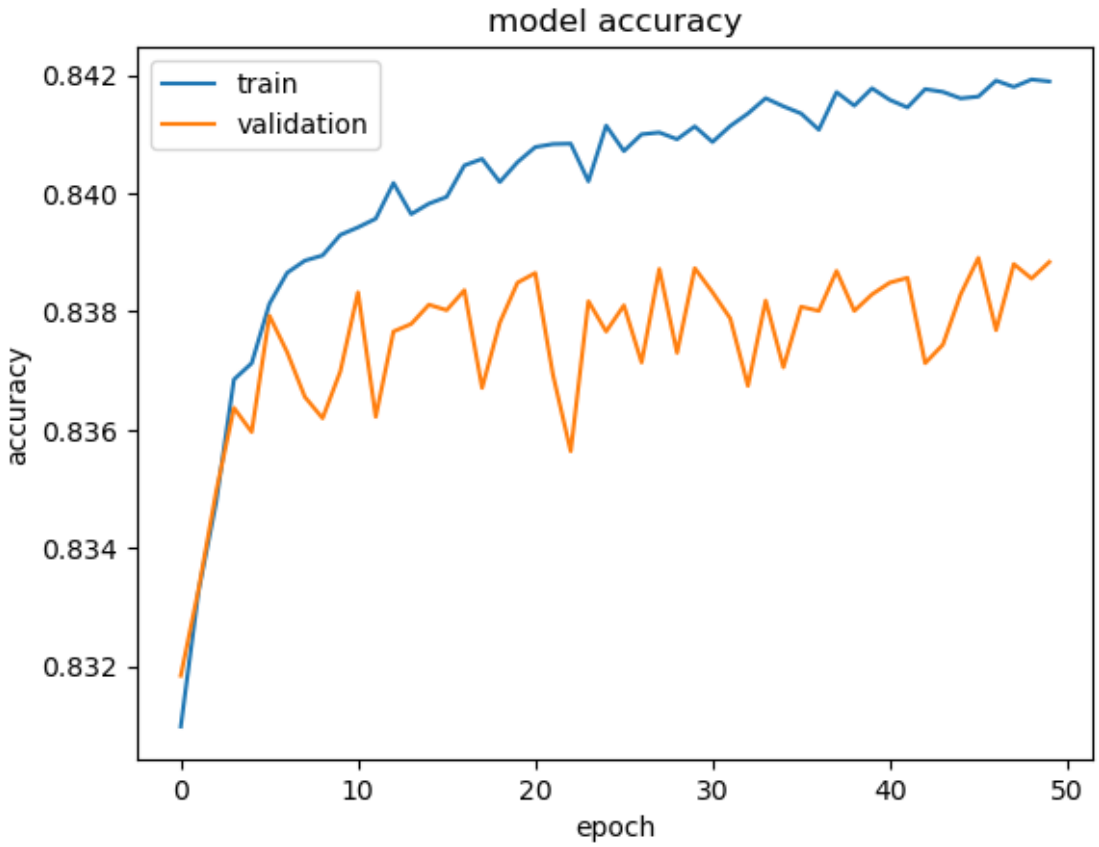

Figure A-8. Historical graph result of the model accuracy over time using the '4.9:1 Class Ratio' iteration of GeneCNN. Accuracy is shown over a period of 50 epochs with both training and validation datasets. Model overfitting is not evident here but there is variance in the validation accuracy once it plateaus at roughly 83.8%.

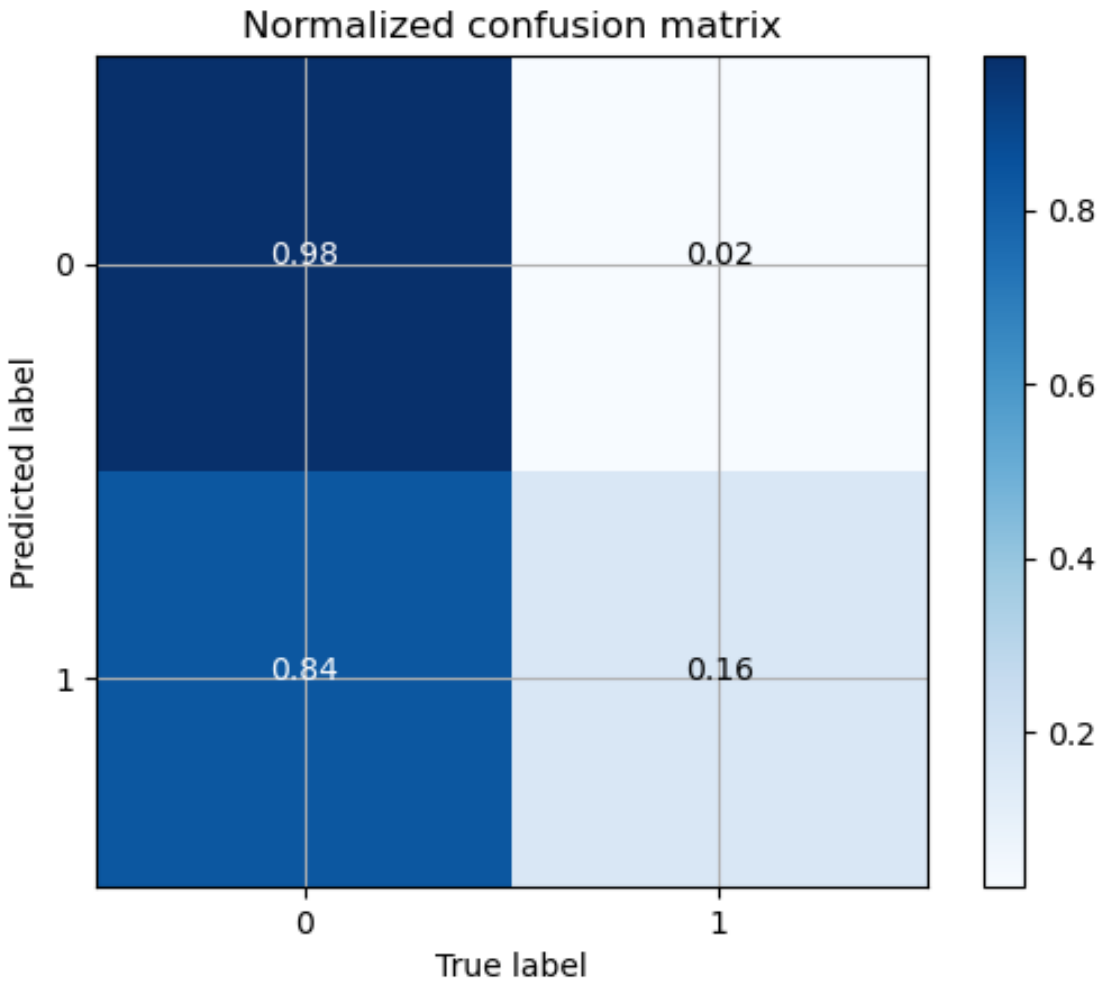

Figure A-9. Historical confusion matrix for the '4.9:1 Class Ratio' iteration of GeneCNN. The nongenic class prediction accuracy was 98%, found in the top left quadrant, while the genic class prediction accuracy was 16%, found in the bottom right quadrant. The false positive rate was 84%, found in the bottom left quadrant, while the false negative rate was 2%, found in the top right quadrant.

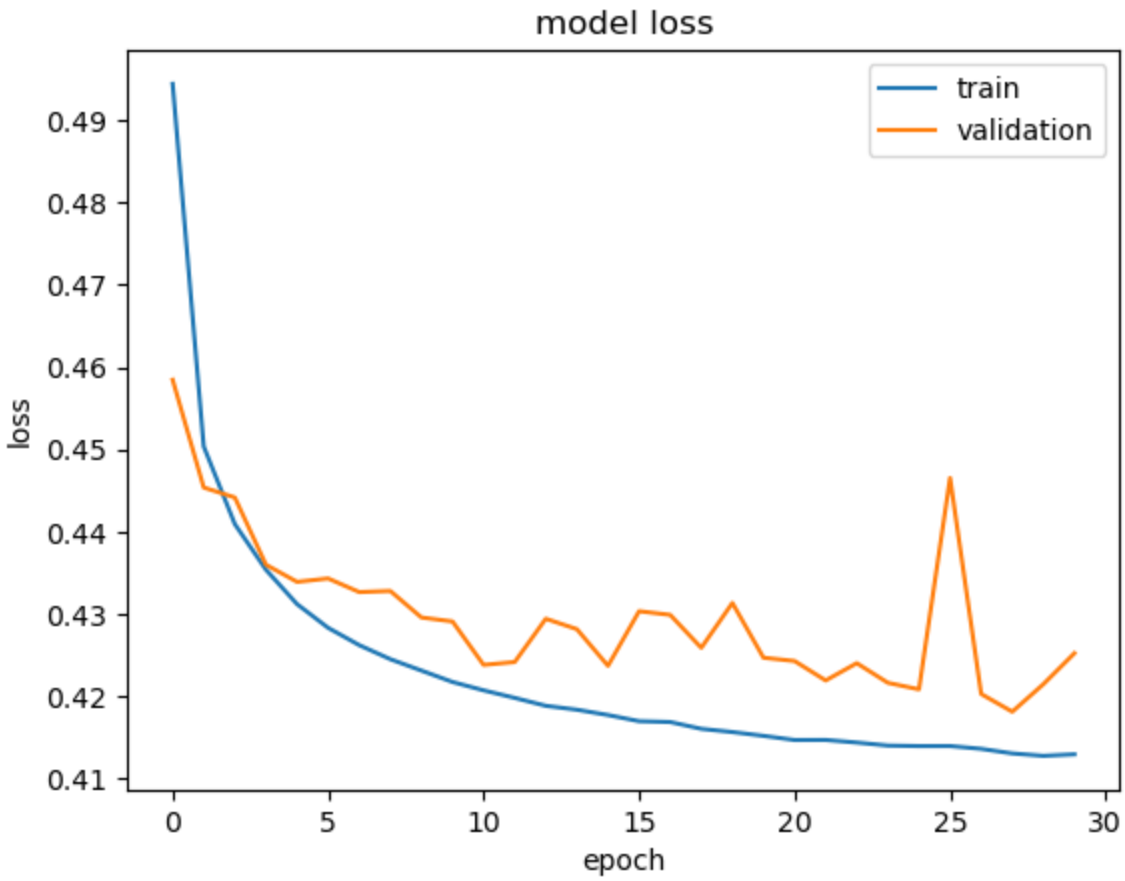

Figure A-10. Historical graph result of the model loss over time using the 'Manual Adjust' iteration of GeneCNN. Loss is shown over a training period of 30 epochs with both training and validation datasets. Model overfitting is not evident here, however there is variance in the validation loss.

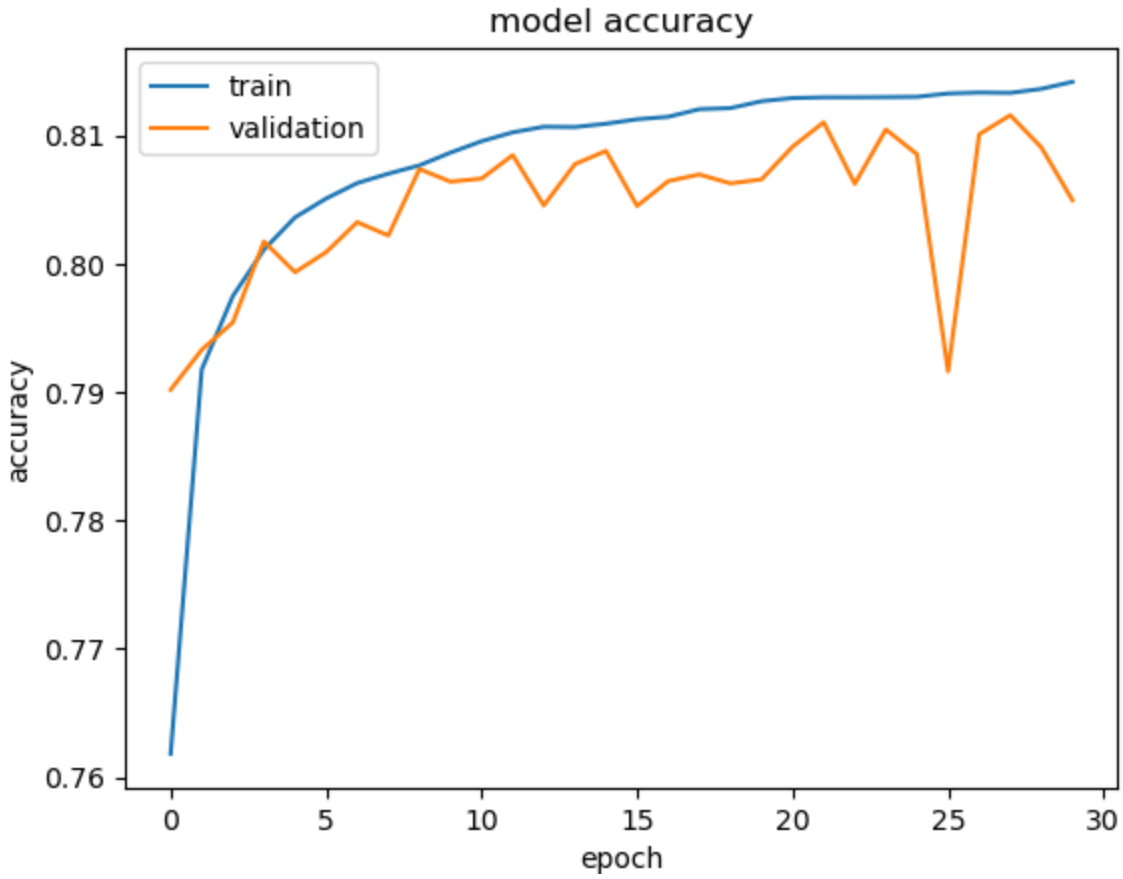

Figure A-11. Historical graph result of the model accuracy over time using the 'Manual Adjust' iteration of GeneCNN. Accuracy is shown over a period of 30 epochs with both training and validation datasets. Model overfitting is not evident here but there is variance in the validation accuracy once it plateaus at roughly 81%.

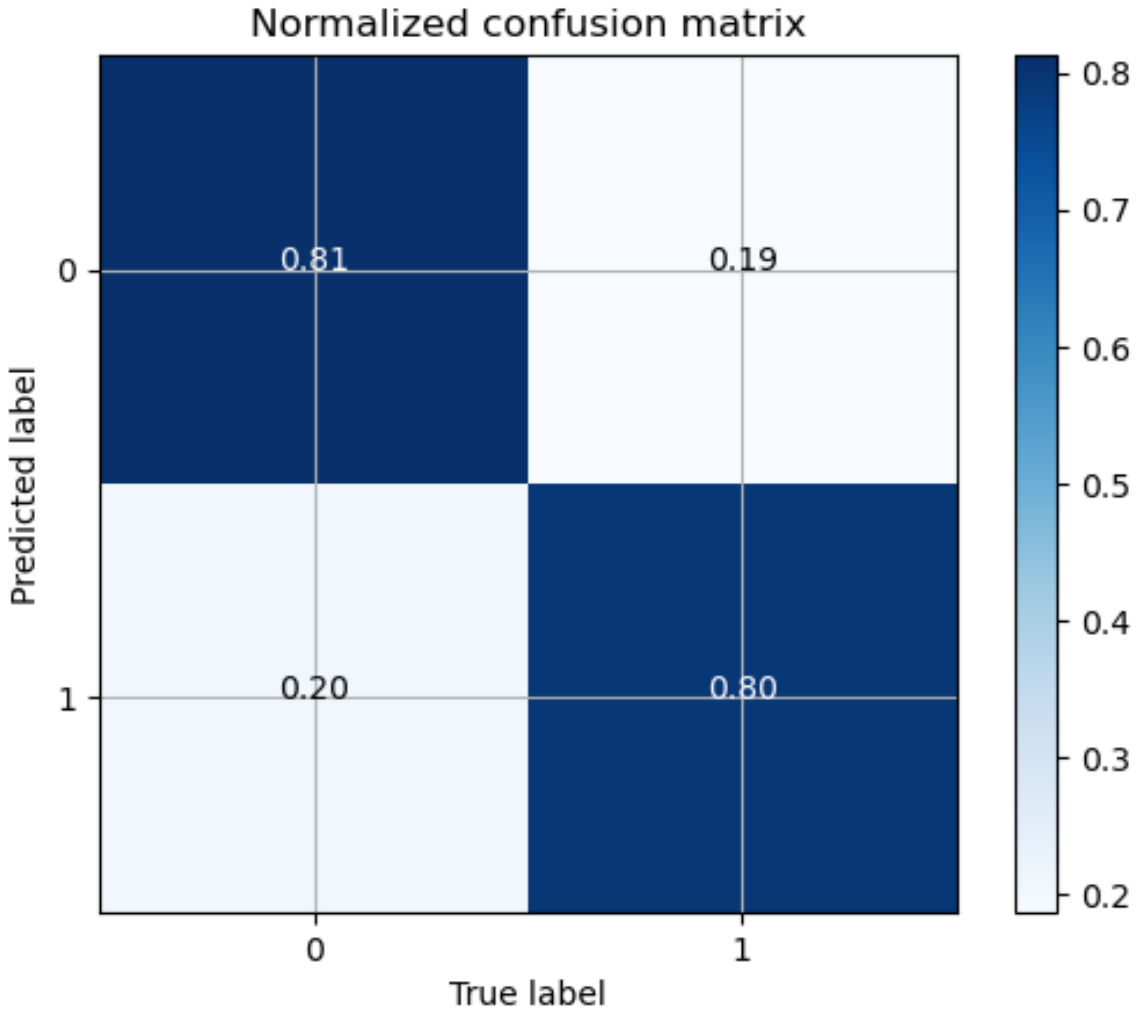

Figure A-12. Historical confusion matrix for the 'Manual Adjust' iteration of GeneCNN. The nongenic class prediction accuracy was 81%, found in the top left quadrant, while the genic class prediction accuracy was 80%, found in the bottom right quadrant. The false positive rate was 20%, found in the bottom left quadrant, while the false negative rate was 19%, found in the top right quadrant.

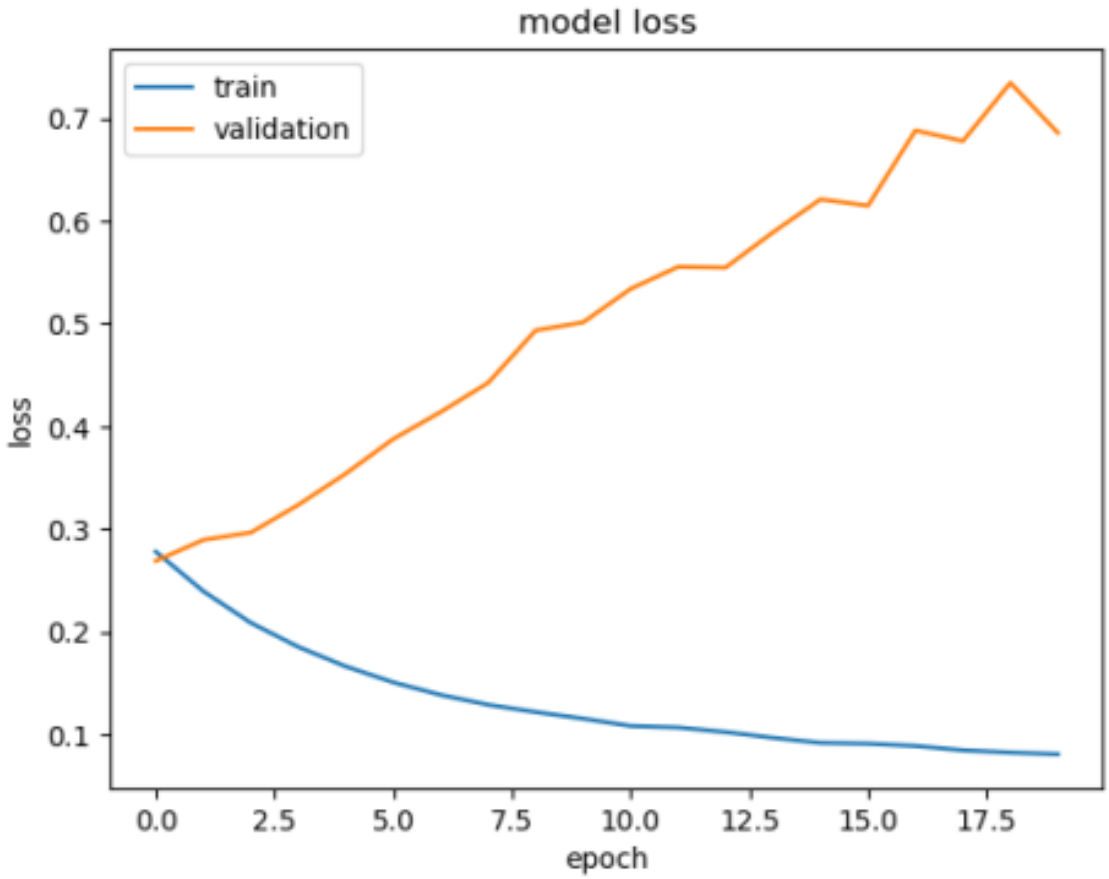

Figure A-13. Historical graph result of the model loss over time using the 'No BN' iteration of GeneCNN. Loss is shown over a training period of 20 epochs with both training and validation datasets. A large degree of model overfitting is present, as the validation loss continued to increase while the training loss decreased.

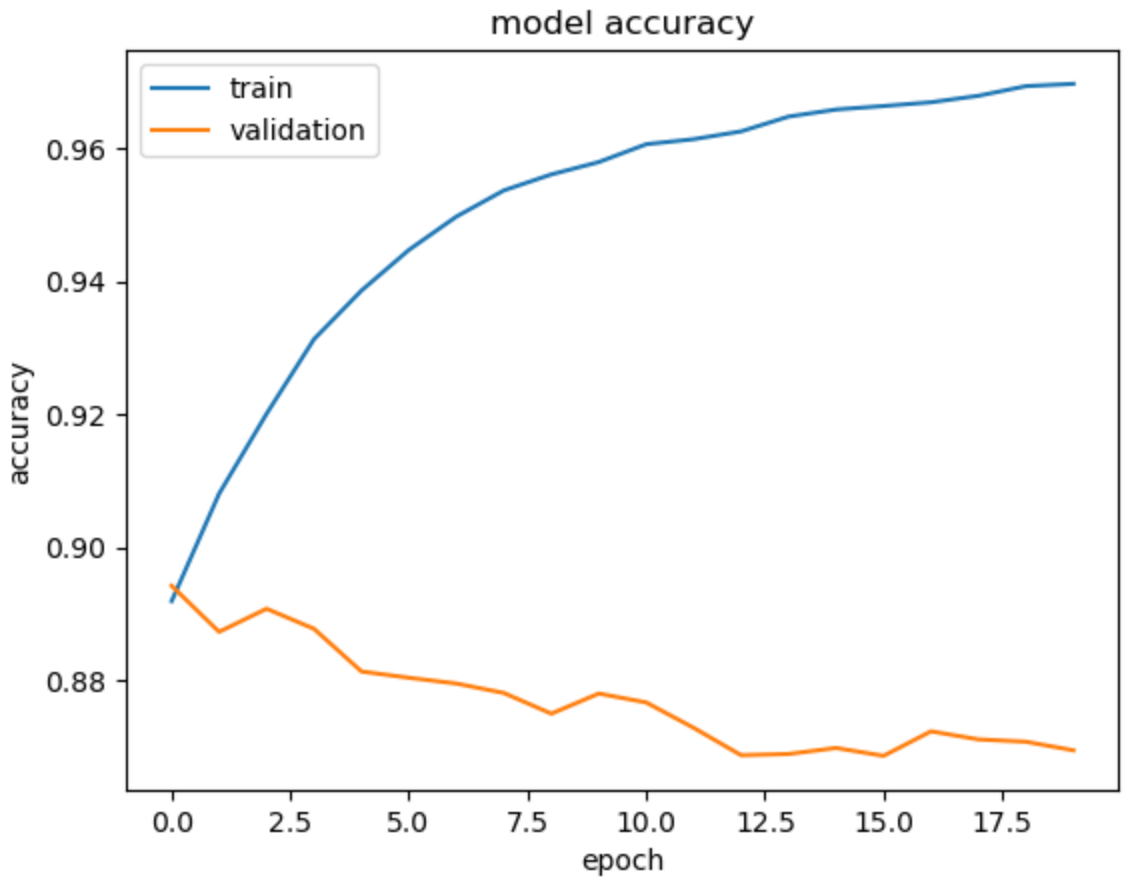

Figure A-14. Historical graph result of the model accuracy over time using the 'No BN' iteration of GeneCNN. Accuracy is shown over a period of 20 epochs with both training and validation datasets. Model overfitting is present as the validation accuracy decreases while the training accuracy increases.

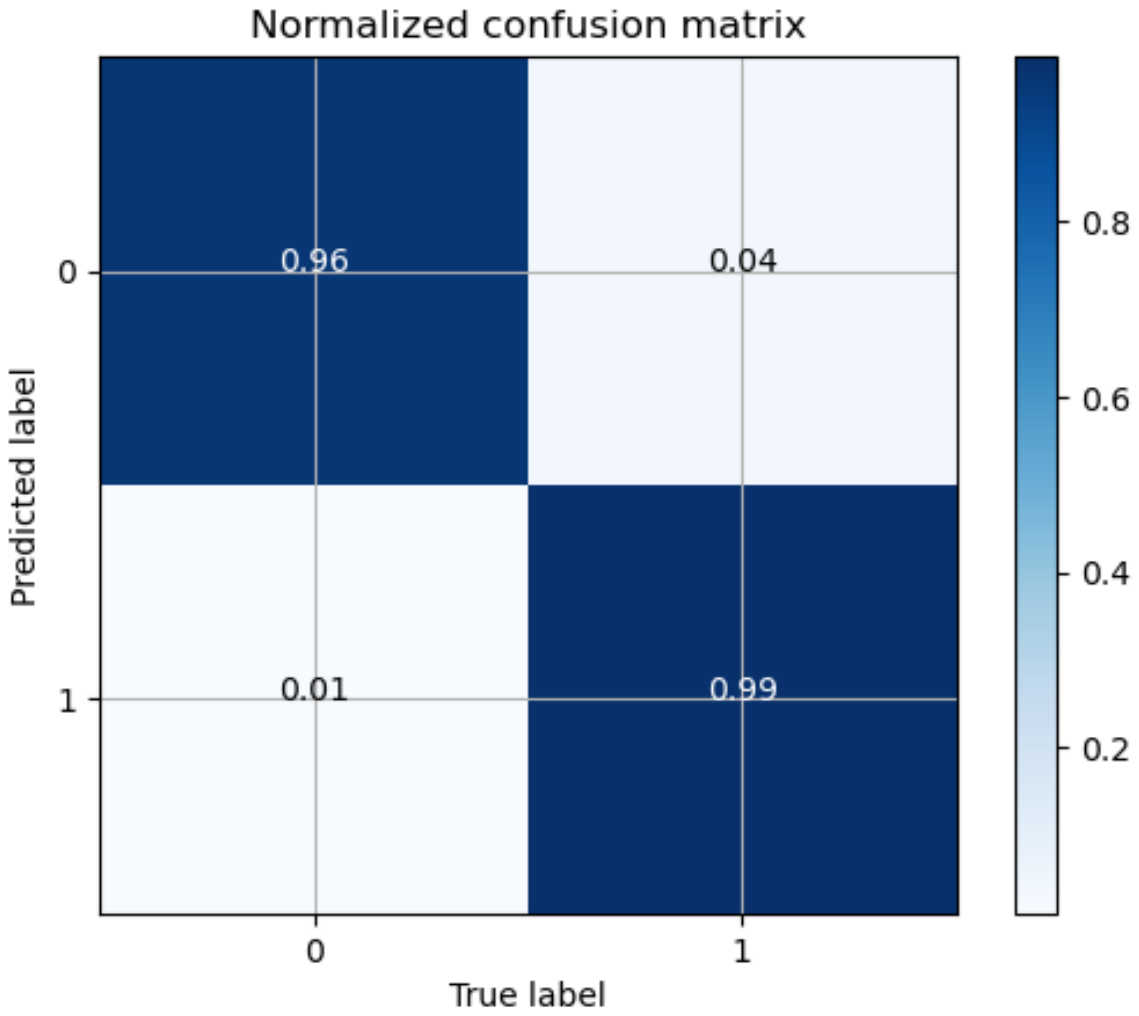

Figure A-15. Historical confusion matrix for the 'No BN' iteration of GeneCNN. The nongenic class prediction accuracy was 96%, found in the top left quadrant, while the genic class prediction accuracy was 99%, found in the bottom right quadrant. The false positive rate was 1%, found in the bottom left quadrant, while the false negative rate was 4%, found in the top right quadrant.